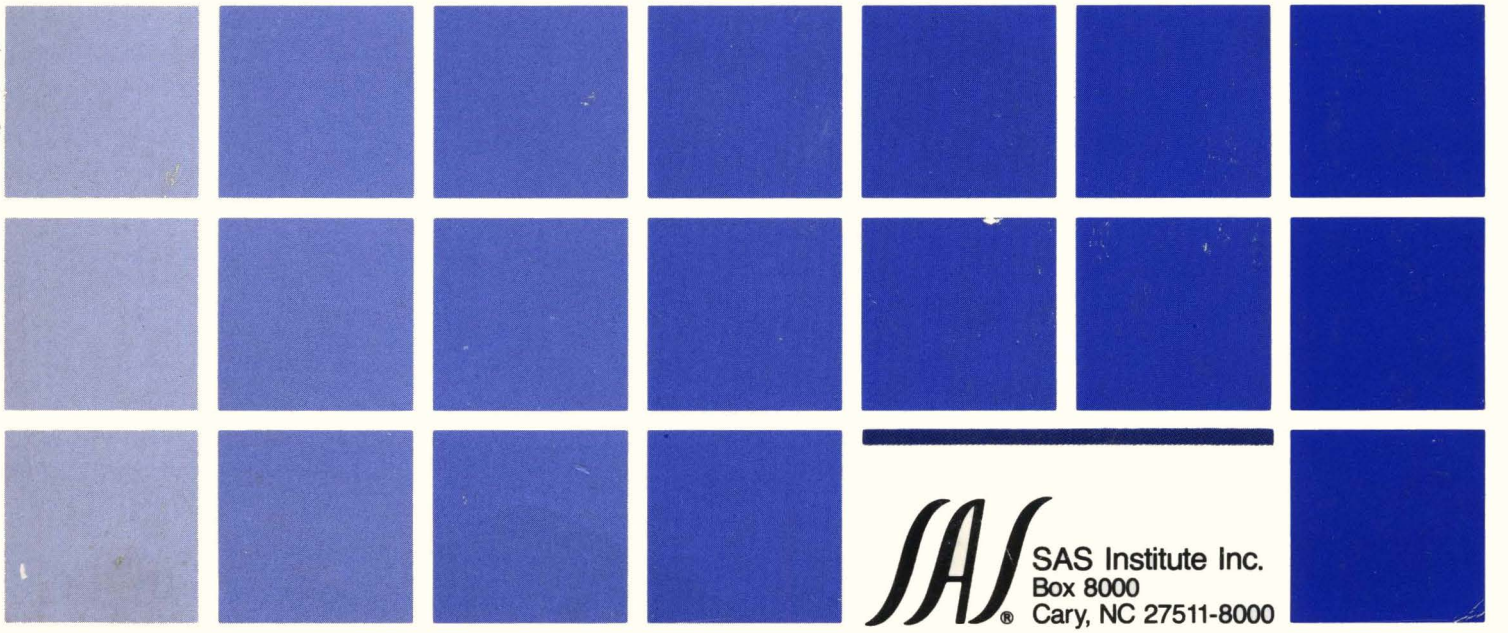
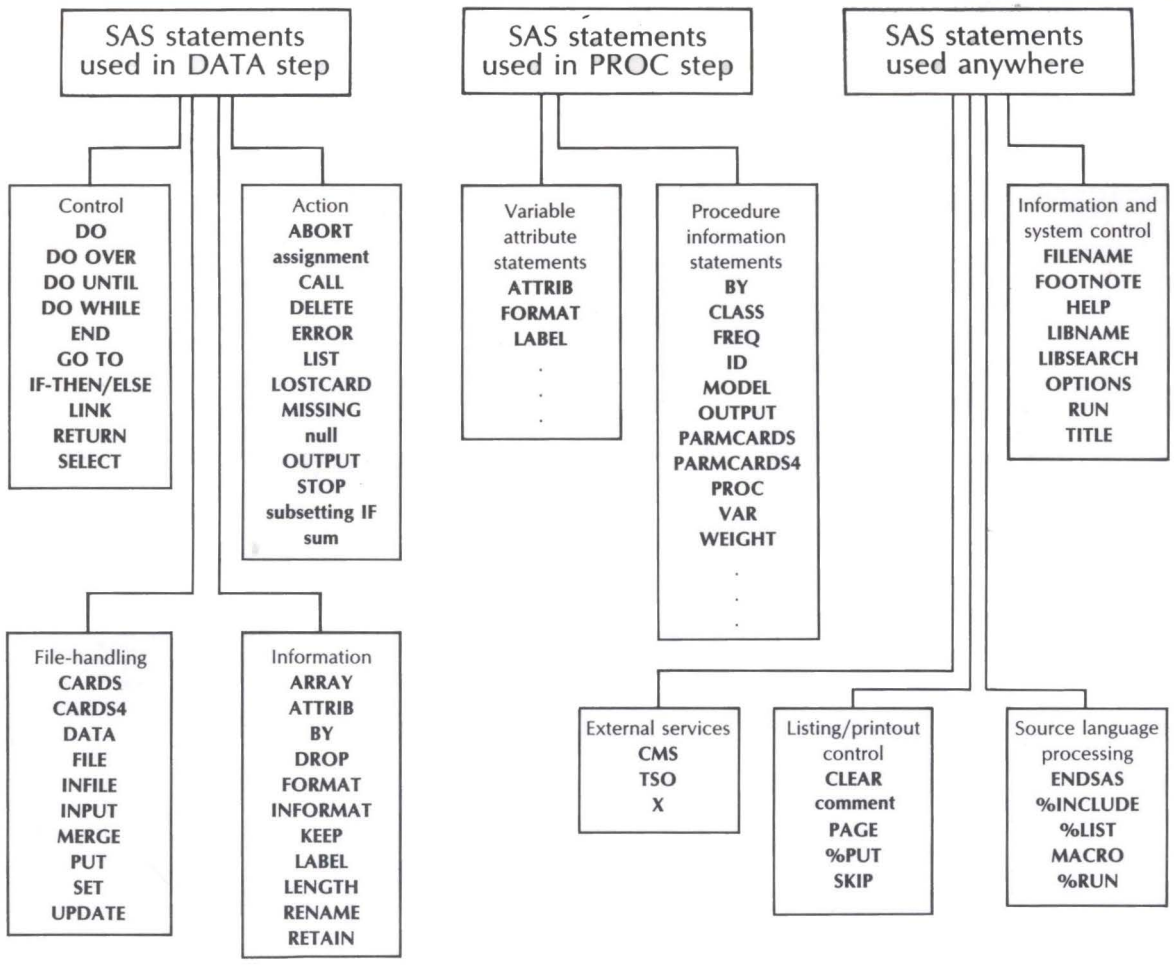
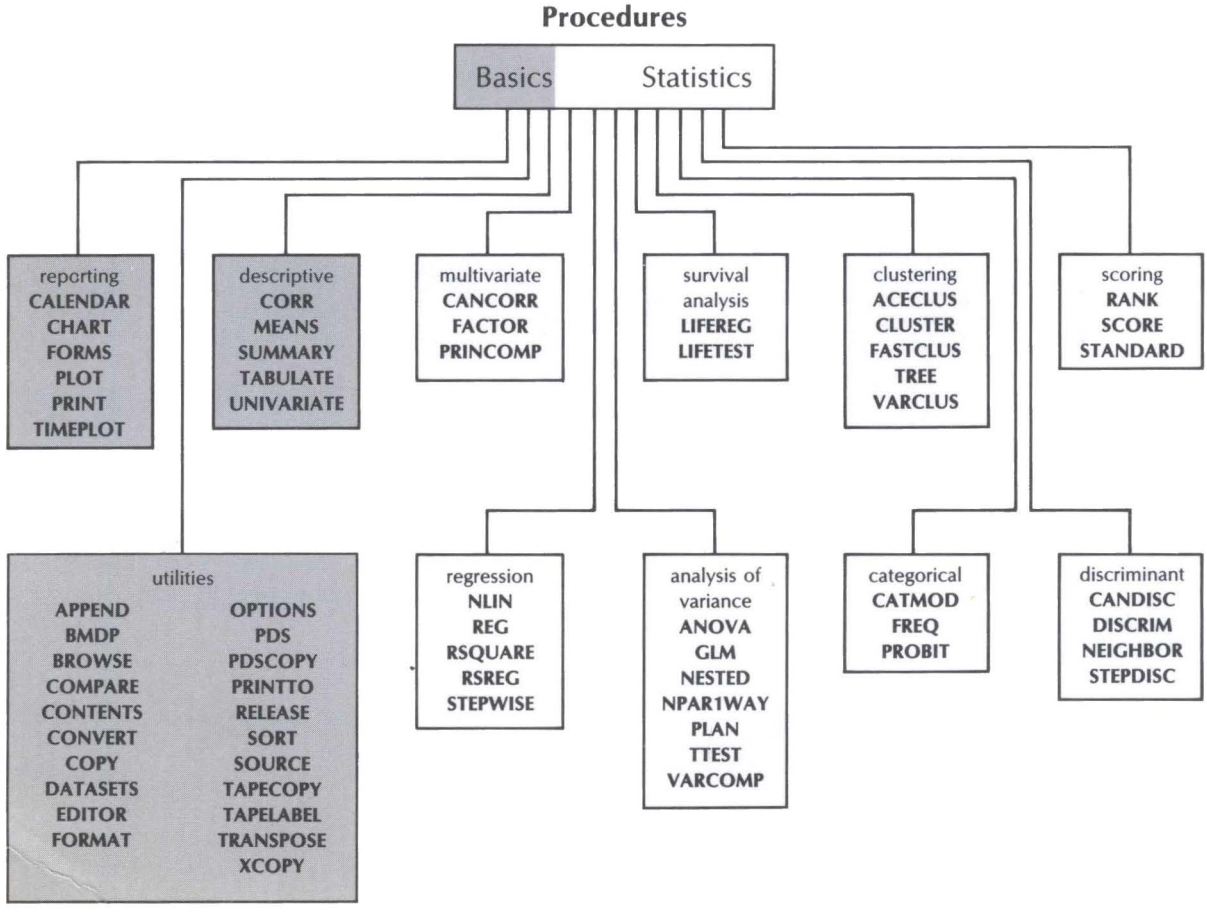
SAS User's Guide: Basics

Version 5 Edition



SAS Institute Inc.
Box 8000
Cary, NC 27511-8000


SAS statements used in DATA step
SAS statements used in PROC step
SAS statements used anywhere
Control
DO
DO OVER
DO UNTIL
DO WHILE
END
GO TO
IF-THEN/ELSE
LINK
RETURN
SELECT
Action
ABORT
assignment
CALL
DELETE
ERROR
LIST
LOSTCARD
MISSING
null
OUTPUT
STOP
subsetting IF
sum
Variable attribute statements
ATTRIB
FORMAT
LABEL
Procedure information statements
BY
CLASS
FREQ
ID
MODEL
OUTPUT
PARMCARDS
PARMCARDS4
PROC
VAR
WEIGHT
Information and system control
FILENAME
FOOTNOTE
HELP
LIBNAME
LIBSEARCH
OPTIONS
RUN
TITLE
File-handling
CARDS
CARDS4
DATA
FILE
INFILE
INPUT
MERGE
PUT
SET
UPDATE
Information
ARRAY
ATTRIB
BY
DROP
FORMAT
INFORMAT
KEEP
LABEL
LENGTH
RENAME
RETAIN
External services
CMS
TSO
X
Listing/printout control
CLEAR
comment
PAGE
%PUT
SKIP
Source language processing
ENDSAS
%INCLUDE
%LIST
MACRO
%RUN


Procedures
Basics
Statistics
reporting
CALENDAR
CHART
FORMS
PLOT
PRINT
TIMEPLOT
descriptive
CORR
MEANS
SUMMARY
TABULATE
UNIVARIATE
multivariate
CANCORR
FACTOR
PRINCOMP
survival analysis
LIFEREG
LIFETEST
clustering
ACECLUS
CLUSTER
FASTCLUS
TREE
VARCLUS
scoring
RANK
SCORE
STANDARD
utilities
APPEND
BMDP
BROWSE
COMPARE
CONTENTS
CONVERT
COPY
DATASETS
EDITOR
FORMAT
OPTIONS
PDS
PDSCOPY
PRINTTO
RELEASE
SORT
SOURCE
TAPECOPY
TAPELABEL
TRANSPOSE
XCOPY
regression
NLIN
REG
RSQUARE
RSREG
STEPWISE
analysis of variance
ANOVA
GLM
NESTED
NPAR1WAY
PLAN
TTEST
VARCOMP
categorical
CATMOD
FREQ
PROBIT
discriminant
CANDISC
DISCRIM
NEIGHBOR
STEPDISC

SAS® User's Guide: Basics, Version 5 Edition

SAS Institute Inc.
Box 8000
Cary, North Carolina 27511-8000

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. *SAS® User's Guide: Basics, Version 5 Edition*. Cary, NC: SAS Institute Inc., 1985. 1290 pp.

SAS® User's Guide: Basics, Version 5 Edition

ISBN 0-917382-65-X

86 85 4 3 2 1

Base SAS® software, the foundation of the SAS System, provides data retrieval and management, programming, statistical, and reporting capabilities. Other products in the SAS System include SAS/FSP,® SAS/GRAPH,® SAS/IMS-DL/I,® SAS/ETS,™ SAS/OR,™ SAS/AF,™ SAS/REPLAY-CICS,™ and SAS/IML™ software. These products are available from SAS Institute Inc., a private company devoted to the support and further development of Institute software and services, including *SAS Communications*,® *SAS Views*,® *SAS Training*,™ and SASware Ballot.™

Contents

Illustrations

Figures

Screens

Tables

Acknowledgments

SAS System for Mainframes

Head: D.D. Cockrell

Supervisor: D.D. Ingold, M.H. Carson, C.L. Garrison, B.R. Hill, P.J. Herold, M.R. Boyd, B.P. Bowman

DATA Step Compiler: D.B. Malkovsky, O.T. Bradley

Display Manager: R.M. Oatsvall, D.B. Malkovsky, D.D. Cockrell

Macro Facility: P.M. Herbert, D.D. Cockrell

SAS System for Minicomputers

Head: R.A. Usanis

Supervisor: S.M. Beatrous, V.M. Dineley, N.L. Blackmon, M.P. Hecht, D.C. Colbert, W.L. Brideson

DATA Step Compiler: B.M. Tindall, T.L. Betancourt, L.M. Hoog, Y. Chen, R.A. Perry, M.V. Schaffer, D.J. Sedlak, S.M. Beatrous, P.F. Busby, R.D. Langston, M.E. Watson, J.A. Polzin

Display Manager: T.N. Lloyd, P.F. Busby

Macro Facility: B.M. Tindall

SAS System under AOS/VS E.Z. Farnum, A.D. Massengill, P.D. Julian, J.A. Polzin, R.K. Whitehead

SAS System under CMS and VM/PC W.F. Miller, W.L. Brideson, D. Meyers, T.P. Hunter, J.C. Weathers, S.F. Tuning, M. Le, J.A. Condrey, K.V. Collins, M.C. Box

SAS System under OS D.D. Cockrell, H.D. Landis, W.F. Heffner, D.C. Berry

SAS System under PRIMOS E.R. Trumbull, A.D. Massengill, P.D. Julian

SAS System under VMS T.B. Cole, P.E. Cregger, M.V. Schaffer

SAS System under VSE O.T. Bradley, D.M. Poll, H.H. Cummings, R.C. Smith

Quality Assurance K.C. Bydalek, B.D. Kernodle, E.C. Brinsfield, T.D. Poole, M.R. Jones, B.J. Carter, J.M. Martin, M.Z. Stokes

Minicomputer Testing A.L. Yang, S.O. Tobacco, L.L. Wharton, M.A. Finch, K.L. Hoffman, G.N. Stenbuck, A.D. Banks

Technical Support M.B. Nichols, M.L. Adair, D.S. Bizzell, D.M. Bravo, V.H. Brocklebank, G.N. Cappy, R.L. Chenoweth, T.L. Clinard, M.J. Cybrynski, S.S. Darden, T.H. Dickey, H.G. Everett, S.A. Gnepp, J.L. Hannah, F.W. Hester, S.J. Jenisch, L.A. Jordan, M.F. Kalt, R.B. Killam, F.E. Lehman, R.M. Ludwick, P.A. McAlister, M.L. Mincey, J.N. Reel, J. Richards, C.E. Routten, M.A. Tesh, T.J. Trott, J.O. Ward, G.G. Wiggs, S.S. Wine, J.B. Yerian

Program authorship includes design, programming, debugging, support, and preliminary documentation.

APPEND	R.D. Langston
BMDP	R.D. Langston
BROWSE	R.D. Langston
CALENDAR	G.K. Howell
CHART	J.H. Goodnight
COMPARE	M.R. Little
CONTENTS	R.D. Langston, K.V. Collins
CONVERT	R.D. Langston
COPY	D.D. Ingold, Mainframe; G.K. Howell, Portable
CORR	D.M. Delong
DATASETS	J.S. Wallace, K.D. Kumar
EDITOR	R.D. Langston
FORMAT	R.D. Langston
FORMS	J.P. Sall
FREQ	W.M. Stanish
MEANS	G.K. Howell
OPTIONS	C.L. Garrison, Mainframe; A.R. Eaton, Portable
PDS	D.D. Ingold
PDSCOPY	O.T. Bradley
PLOT	J.H. Goodnight
PRINT	R.D. Langston

PRINTTO	D.D. Ingold
RELEASE	D.D. Ingold
SORT	D.D. Ingold, Mainframe; J.H. Goodnight, Portable
SOURCE	O.T. Bradley
SUMMARY	G.K. Howell
TABULATE	A.R. Eaton
TAPECOPY	D.D. Cockrell, J.C. Weathers
TAPELABEL	D.D. Ingold, T.P. Hunter
TIMEPLOT	R.D. Langston
TRANSPOSE	W.S. Sarle
UNIVARIATE	D.M. Delong
XCOPY	G.K. Howell

SAS Institute Inc. is unusually fortunate to have had many people make significant and continuing contributions to its development.

First among them are the chairpersons and other active members of the SAS Users Group International (SUGI). Julian Horwich, now of Arnar-Stone Laboratories, set up the first regional meetings at Abbott Laboratories in North Chicago in 1975, and organized and chaired the first international SUGI meeting in Lake Buena Vista, Florida, in 1976. Dr. Ronald W. Helms of the University of North Carolina at Chapel Hill chaired the second annual conference, held in New Orleans in 1977; Kenneth Offord of the Mayo Clinic was program chairman.

The 1978 SUGI conference in Las Vegas was co-chaired by Dr. Rodney Strand and Dr. Michael Farrell of Oak Ridge National Laboratories. Dr. Ramon C. Littell of the University of Florida and Dr. William Wilson of the University of North Florida co-chaired the 1979 SUGI meeting in Clearwater, Florida. The 1980 SUGI conference in San Antonio, Texas, was chaired by Rudolf J. Freund of Texas A&M University. Kenneth L. Koonce of Louisiana State University chaired the 1981 SUGI conference held in Lake Buena Vista, Florida. The 1982 conference held in San Francisco, California, was chaired by Helene Cavior of the Bureau of Prisons. In 1983 SUGI was in New Orleans with J. Philip Miller of Washington University School of Medicine as chairman. Hollywood Beach, Florida was the site of the 1984 SUGI conference chaired by Sally Carson of the Rand Corporation. Rodney Strand, Science Applications, Inc., and Mike Farrell of ORNL co-chaired SUGI in Reno, Nevada in 1985.

The University Statisticians of the Southern Experiment Stations have made numerous contributions over the years. We are especially indebted to:

Wilbert P. Byrd	Clemson University
Richard Cooper	USDA
R.J. Freund	Texas A & M University
Charles Gates	Texas A & M University
Don Henderson	ORI
David Hurst	University of Alabama at Birmingham
Kenneth Koonce	Louisiana State University
Clyde Y. Kramer (deceased)	Virginia Polytechnic Institute, University of Kentucky, and Upjohn Laboratories
Robert D. Morrison	Oklahoma State University
Richard M. Patterson	Auburn University

William L. Sanders	University of Tennessee
Glenn Ware	University of Georgia
T.J. Whatley	University of Tennessee.

We are deeply grateful to our good neighbor, the Department of Statistics at North Carolina State University, and especially to these staff members:

David D. Mason	Chairman (retired)
Jolayne Service	(now at the University of California, Irvine)
Sandra Donaghy	
Carroll Perkins	(now with Westinghouse)
Francis J. Verlinden	
Evelyn Wilson.	

We also wish to express our sincere gratitude to the following people who have contributed in many different ways to the success of the SAS System. Others who made contributions to the statistical procedures are recognized in *SAS User's Guide: Statistics*:

Ray Barnes	Upjohn Pharmaceutical Co.
Mark Bercov	Gulf Canada
Dick Blocker	Washington University
George Chao	Arnar-Stone Laboratories
Daniel Chilko	University of West Virginia
Ray Danner	National Institute of Health
Paul Fingerman	Boeing Computer Services
Terry Flynn	Tymshare
Michael Foxworth	University of South Carolina
Harold Gugel	General Motors Corporation
Donald Guthrie	University of California at Los Angeles
James Guthrie	Ameritrust
Frank Harrell	Duke University
Loren Harrell	Business Information Technology Inc.
Jim Harrington	Monsanto Research
Walter Harvey	Ohio State University
Ronald Helms	University of North Carolina at Chapel Hill
Harold Huddleston	Data Collection & Analysis, Inc., Falls Church, Virginia
Emilio A. Icaza	Louisiana State University
Ruth Ingram	Proctor & Gamble
William Kennedy	Iowa State University
Melvin Klassen	University of Victoria
John Klinkner	American Republic Insurance Company
H.W. (Barry) Merrill	Merrill Consultants
J. Philip Miller	Washington University
Mario Morino	Morino Associates Inc.
Curt Mosso	University of California at Santa Barbara
Richard Nelson	Clemson University
Robert Parks	Washington University

Virginia Patterson	University of Tennessee
Jim Penny	Academic Computer Center University of North Carolina at Greensboro
Pete Rikard	Virginia Commonwealth University
John Ruth	University of Toronto
Robert Schechter	Scott Paper Company
Roger Smith	USDA
Robert Teichman	ICI Americas Inc.
Jim Walker	Morino Associates Inc.

Two of the founders of SAS Institute Inc. deserve special mention.

The present SAS documentation can be traced to the work of Jane T. Helwig, now at Seasoned Systems Inc., Chapel Hill, N.C., who gathered and wrote much of the material on which this manual is based.

Finally, SAS Institute and all SAS users owe a debt of gratitude to Anthony J. Barr, Barr Systems, Raleigh, N.C. His work on the SAS supervisor and compiler, as well as the procedures that he wrote, help make the SAS System the useful tool it is today.

The final responsibility for the SAS System lies with SAS Institute alone. We hope that you will always let us know your feelings about the system and its documentation. It is through such communications that the progress of SAS software has been accomplished.

The Staff of SAS Institute Inc.

Preface

This volume, the *SAS User's Guide: Basics, Version 5 Edition*, contains the primary description of the SAS language and base SAS procedures for data processing, summarizing, and reporting. Advanced statistical procedures in base SAS software are documented in the *SAS User's Guide: Statistics, Version 5 Edition*.

What is the SAS System?

The SAS System is a software system for data analysis. The goal of SAS Institute is to provide data analysts one system to meet all their computing needs. When your computing needs are met, you are free to concentrate on results rather than on the mechanics of getting them. Instead of learning programming languages, several statistical packages, and utility programs, you only need to learn the SAS System.

To the all-purpose base SAS software, you can add tools for graphics, forecasting, data entry, and interfaces to other data bases to provide one total system. SAS software runs on IBM 370/30xx/43xx and compatible machines in batch and interactively under OS and TSO, CMS, VSE, and SSX; on Digital Equipment Corporation VAX™ 11/7xx series under VMS;™ Data General ECLIPSE® series under AOS/VS; MV series under AOS/VS; Prime Series 50 under PRIMOS;® and IBM PC AT/370 and XT/370 under VM/PC. Note: not all products are available for all operating systems.

Base SAS software provides tools for:

- information storage and retrieval
- data modification and programming
- report writing
- statistical analysis
- file handling.

Information storage and retrieval The SAS System reads data values in virtually any form from cards, disk, or tape and then organizes the values into a SAS data set. The data can be combined with other SAS data sets using the file-handling operations described below. The data can be analyzed statistically and can be used to produce reports. SAS data sets are automatically self-documenting since they contain both the data values and their descriptions. The special structure of a SAS data library minimizes maintenance.

Data modification and programming A complete set of SAS statements and functions is available for modifying data. Some program statements perform standard operations such as creating new variables, accumulating totals, and checking for errors; others are powerful programming tools such as DO/END and IF-THEN/ELSE statements. The data-handling features are so valuable that base SAS software is used by many as a data base management system.

Report writing Just as base SAS software reads data in almost any form, it can write data in almost any form. In addition to the preformatted reports that SAS

VAX™ and VMS™ are trademarks of Digital Equipment Corp., Maynard, MA, USA.

Eclipse® is the registered trademark of Data General Corp., Westboro, MA, USA.

PRIMOS® is the registered trademark of Prime Computer, Inc., Framingham, MA, USA.

procedures produce, SAS software users can design and produce printed reports in any form, as well as punched cards and output files.

Statistical analysis The statistical analysis procedures in the SAS System are among the finest available. They range from simple descriptive statistics to complex multivariate techniques. Their designs are based on our belief that you should never need to tell the SAS System anything it can figure out by itself. Statistical integrity is thus accompanied by ease of use. Especially noteworthy statistical features are the linear model procedures, of which GLM (**G**eneral **L**inear **M**odels) is the flagship.

File handling Combining values and observations from several data sets is often necessary for data analysis. SAS software has tools for editing, subsetting, concatenating, merging, and updating data sets. Multiple input files can be processed simultaneously, and several reports can be produced in one pass of the data.

Other SAS System Products

With base SAS software, you can integrate SAS software products for graphics, data entry, operations research, and interfaces to other data bases to provide one total system:

- SAS/FSP software—interactive, menu-driven facilities for data entry, editing, retrieval of SAS files, letter writing, and spreadsheet analysis
- SAS/GRAPH software—device-intelligent color graphics for business and research applications
- SAS/REPLAY-CICS software—interface that allows users of CICS/OS/VS and CICS/DOS/VS to store, manage, and replay SAS/GRAPH displays
- SAS/OR software—decision support tools for operations research and project management
- SAS/AF software—a full-screen, interactive applications facility
- SAS/ETS software—expanded tools for business analysis, forecasting, and financial planning
- SAS/IMS-DL/I software—interface for reading, updating, and writing IMS/VS or CICS DL/I data bases
- SAS/IML software—multi-level, interactive programming language whose data elements are matrices.

SAS Institute Documentation

Using this manual To serve both as a reference for experienced SAS users and as a learning tool for new users, this manual is organized to reflect the structure of the SAS language. All SAS programs are composed of single DATA or PROC steps or combinations of DATA and PROC steps. Similarly, the information in this manual is arranged according to where it can be used in a SAS program—in a DATA step, in a PROC step, or anywhere.

Introductory chapters The first chapter, "Getting Started," includes some things you need to know about your own computer environment before you can begin. This discussion will help you decide which parts of this manual you need to read right away. The next chapter is an overview of the SAS language that describes the basics of SAS and defines the terms used throughout SAS literature.

The next section describes the DATA step and includes an introduction, statements used in the DATA step (in alphabetical order), SAS expressions, SAS functions, and a chapter on DATA step applications. The PROC step is described in the next section with an introductory chapter, the statements used in the PROC step (in alphabetical order), and some procedure step applications.

The next major section gives an introduction to the SAS System and includes the statements used anywhere in a SAS job (in alphabetical order), chapters describing SAS log and procedure output, SAS display manager, SAS informats and formats, missing values, SAS files, SAS system options, and the SAS macro language.

Procedure descriptions follow alphabetically in the next chapters. Each procedure description is self-contained; you need to be familiar with only the most basic features of the SAS System and SAS terminology to use most procedures. The statements and syntax necessary to run each procedure are presented in a uniform format throughout the manual.

Each procedure description is divided into the following major parts:

ABSTRACT: a short paragraph describing what the procedure does.
INTRODUCTION: introductory and background material, including definitions and occasional introductory examples.
SPECIFICATIONS: reference section for the syntax of the control language for the procedure.
DETAILS: expanded descriptions of features, internal operations, output, treatment of missing values, computational methods, required computational resources, and usage notes.
EXAMPLES: examples using the procedure, including data, SAS statements, and printed output. You can reproduce these examples by copying the statements and data and running the job.
REFERENCES: a selected bibliography.
NOTES: a listing of operating system differences.

There are three appendices, including a summary of Version 5 changes and enhancements to base SAS software, operating system notes, and a discussion of full-screen editing capabilities with the SAS System.

If you have any problems with this manual, please take time to complete the review page at the end of this book and send it to SAS Institute. We will consider your suggestions for future editions. In the meantime, ask your installation's SAS Software Consultant for help.

New features in Version 5 base SAS software Version 5 of base SAS software contains many new features, including SAS display manager, a facility allowing you to interact with all parts of your SAS job from a full-screen terminal. Among SAS language enhancements are explicitly subscripted arrays, the ability to select blocks of code for execution, and specification of variables' format, informat, label, and length in one statement. Base SAS software procedures in Version 5 include many new features. For example, one new procedure, PROC COMPARE, compares variable values in two data sets and reports differences. The XCOPY procedure reformats a SAS data library under a mainframe operating system for use under a minicomputer operating system.

SAS release? To find out which release of SAS software you are using, run any SAS job and look at the release number in the notes at the beginning of the SAS log. This user's guide documents Version 5 base SAS software. If you have an earlier release (for example, the 82.4 release), you should use the 1982 edition of the *SAS User's Guide: Basics*.

Other SAS Institute manuals and technical reports Below is a list of other manuals that document Version 5 SAS System software:

SAS User's Guide: Statistics, Version 5 Edition
SAS/GRAPH User's Guide, Version 5 Edition
SAS/FSP User's Guide, Version 5 Edition

SAS/OR User's Guide, Version 5 Edition
SAS/AF User's Guide, Version 5 Edition
SAS/ETS User's Guide, Version 5 Edition
SAS/IML User's Guide, Version 5 Edition

The SAS Technical Report Series documents work in progress, describes new supplemental procedures, and covers a variety of applications areas. Some of the features described in these reports are still in experimental form and are not yet available as SAS procedures.

Write to SAS Institute for a current publications catalog, which describes the manuals as well as technical reports and lists their prices.

SAS Services to Users

Technical support SAS Institute supports users through the Technical Support Department. If you have a problem running a SAS job, you should contact your site's SAS Software Consultant. If the problem cannot be resolved locally, your local support personnel should call the Institute's Technical Support Department at (919) 467-8000 on weekdays between 9:00 a.m. and 5:00 p.m. Eastern Standard Time. A brochure describing the services provided by the Technical Support Department is available from SAS Institute.

Training SAS Institute sponsors a comprehensive training program, including programs of study for novice data processors, statisticians, applications programmers, systems programmers, and local support personnel. *SAS Training*, a semiannual training publication, describes the total training program and each course currently being offered by SAS Institute.

News magazine *SAS Communications* is the quarterly news magazine of SAS Institute. Each issue contains ideas for more effective use of the SAS System, information about research and development underway at SAS Institute, the current training schedule, new publications, and news of the SAS Users Group International (SUGI).

To receive a copy of *SAS Communications* regularly, send your name and complete address to:

SAS Institute Mailing List
SAS Institute Inc.
Box 8000
Cary, NC 27511-8000

Sample library One of the data sets included on the base SAS software installation tape is called SAS.SAMPLE. This data set contains sample SAS applications to illustrate features of SAS procedures and creative SAS programming techniques that can help you gain an in-depth knowledge of SAS capabilities.

Here are a few examples of programs included:

ANOVA	analyzing a Latin-square split-plot design
ARIMA5	fitting an intervention model to an ozone time series
CENSUS	reading hierarchical files of the U.S. Census Bureau Public Use Sample tapes
HARRIS	reading Harris Poll tapes coded in column-binary format
PDL	fitting a polynomial distributed lag regression model
TEACH	teaching arithmetic to your child.

Check with your SAS Software Consultant to find out how to access the library since it may have been put on disk at your site.

SUGI

The SAS Users Group International (SUGI) is a nonprofit association of professionals who are interested in how others are using the SAS System. Although SAS Institute provides administrative support, SUGI is independent from the Institute. Membership is open to all users at SAS sites, and there is no membership fee.

Annual conferences are structured to allow many avenues of discussion. Users present invited and contributed papers on various topics, for example:

- computer performance evaluation and systems software
- econometrics and time series
- graphics
- information systems
- interactive techniques
- statistics
- tutorials in SAS System software.

Proceedings of the annual conferences are distributed free to SUGI registrants. Extra copies may be purchased from SAS Institute.

SASware Ballot SAS users provide valuable input toward the direction of future SAS development by ranking their priorities on the annual SASware Ballot. The top vote-getters are announced at the SUGI conference. Complete results of the SASware Ballot are also printed in the *SUGI Proceedings*.

Supplemental library SAS users at many installations have written their own SAS procedures for a wide variety of specialized applications. Some of these user-written procedures are available through the SUGI supplemental library and are documented in the *SUGI Supplemental Library User's Guide*. The procedures in the supplemental library are sent to each installation that licenses base SAS software, although only a few procedures are supported by SAS Institute staff.

Licensing the SAS System

The SAS System is licensed to customers in the Western Hemisphere from the Institute's headquarters in Cary, NC. To serve the needs of our international customers, the Institute maintains subsidiaries in the United Kingdom, New Zealand, Australia, Singapore, Germany, and France. In addition, agents in other countries are licensed distributors for the SAS System. For a complete list of offices, write or call:

SAS Institute Inc.
SAS Circle
Box 8000
Cary, NC 27511-8000
(919) 467-8000

USING THE SAS® SYSTEM

Getting Started: An Introduction for the Novice

Introduction to the SAS® Language

Chapter 1

Getting Started: An Introduction for the Novice

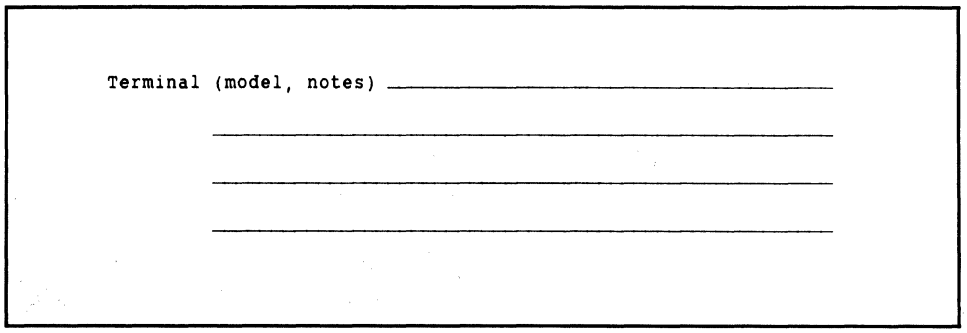
INTRODUCTION

The SAS System works with many different arrangements of computer hardware and software. Since this book applies generally to a wide range of systems, you will have to discover certain facts about your computing environment on your own. If you are already experienced with the computer on which you plan to execute SAS programs, you can skip this chapter. Otherwise, you can begin using the SAS System only after you know the answers to the questions in this chapter. By finding these answers, you will quickly get the vocabulary necessary for this manual. If you need assistance, your local SAS Software Consultant is a good source of information. (Fill in the blanks for future reference.)

```
SAS Software Consultant _____________________Phone _____________

Other Systems Personnel _____________________ Phone _____________

                        _____________________ Phone _____________

                        _____________________ Phone _____________
```

STARTING WITH QUESTIONS

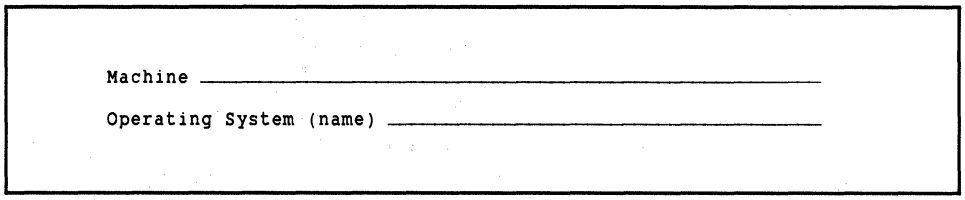
What kind of input device will you use to communicate with your computer? One type of input device is a computer terminal with a keyboard for entering text and commands. Some terminals print on paper rolls; others have screens where text is displayed.



In some instances what you can do with SAS software depends on the capabilities of your input device. For this reason, you must master the basics of the input method you plan to use before you try to master SAS programming. For example, knowing the functions of your terminal's keys is essential from the beginning.

Another method is to enter a SAS program on keypunched cards; the input device is a card reader attached to the computer.

What is the name of the operating system that executes the SAS System on your computer? The *operating system* is the software that controls the activities of the computer, including access to the SAS System. You will use the operating system's control language to request SAS software.



Learn the name of your operating system at this point. Follow this manual's references to the "Operating System Notes" appendix and other documentation as necessary. Specific information about operating systems is also given in **Notes** throughout the book.

Which is the best execution mode available to you for your task? *Execution mode* describes the specific way you submit programs to be processed. The execution mode you choose depends on the capabilities of your input device, the alternatives available with your operating system, and what you want the SAS System to do.

Generally, if you have a large amount of data that are already entered, you will be able to consider *batch mode*; for example, a monthly payroll is often processed in batch. Programs on keypunched cards are executed in batch mode, and batch mode is available from terminals on some operating systems. If you must have information processed as soon as it is entered, you work from a terminal in *interactive mode*. Airline flight reservation systems and automatic teller transactions at banks require interactive mode, also called *conversational mode*.

```
Execution Mode_____________________________________________
```

Given your operating system's batch or interactive modes, which SAS modes of execution are available to you? There are four execution modes available with the SAS System as shown in **Figure 1.1**, although all modes may not be available locally on your operating system. You can choose different execution modes for different SAS applications.

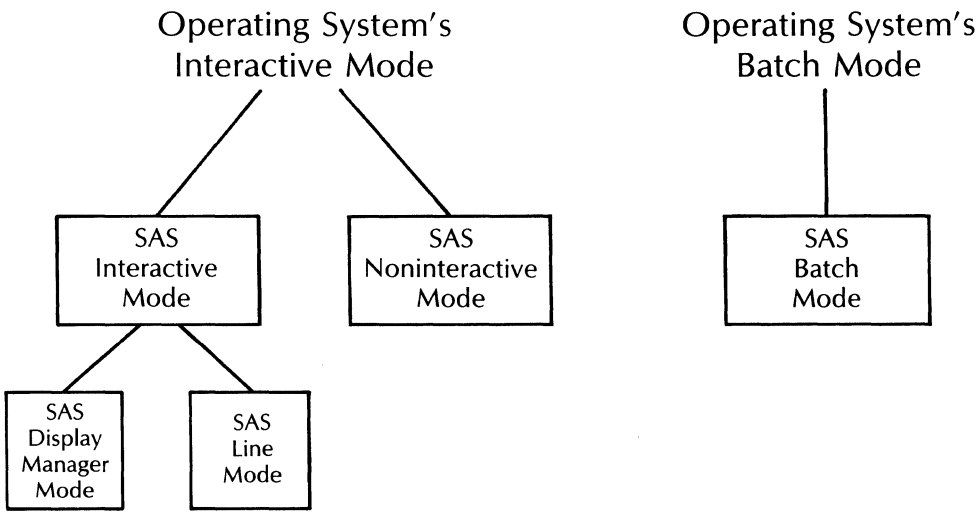


Figure 1.1 Four SAS Execution Modes

A *batch SAS job* is submitted to the computer and waits its turn for execution along with the jobs of other users on the operating system. If you are using a terminal, you prepare a file containing statements in the operating system's control language and all the SAS program statements for the job, or you can prepare a similar deck of keypunched cards to be read.

A *noninteractive SAS session* also requires you to prepare a file containing SAS program statements without control statements; however, this job is submitted to the system for immediate rather than batch execution.

During an *interactive line-mode SAS session*, you enter SAS statements one or several at a time, and the system responds to each statement.

If you have a terminal on which the SAS System supports full-screen capability, you can take advantage of the features of an *interactive SAS display manager session*. SAS display manager is a set of screens and special commands for entering SAS programs and viewing results simultaneously. Among other advantages, the display manager method combines the best features of interactive line mode and noninteractive mode.

Before you access SAS software, you need to decide which execution methods are available and which one best suits your immediate needs. The SAS Software Consultant will be aware of any local requirements or restrictions.

```
SAS Execution Modes  (check if available)
_____  SAS Batch Mode
_____  SAS Noninteractive Mode
_____  SAS Line Mode
_____  SAS Display Manager
```

The four modes of execution are described for different operating systems in the "Operating System Notes" appendix. See the "SAS Display Manager" chapter for details on using this method.

Which text editor available on your operating system will you use to prepare files for SAS programs? With SAS batch mode from a terminal and SAS noninteractive mode you must learn how to type program statements into a file on your operating system using an *editor*. (The SAS display manager has a built-in editor.)

An editor is a set of commands, some possibly assigned to terminal keys, for entering and changing text. (An optional use of SAS line mode and SAS display manager also requires file preparation.) Ask your SAS Software Consultant or operating system personnel to recommend a text editor available on your operating system. Refer to specific documentation or an on-line tutorial and practice using the editor before attempting to prepare SAS program statements.

```
Editor _____________________________________________

Access instructions _____________________________________

                    _____________________________________

                    _____________________________________
```

FINDING THE SAS SYSTEM

How do you get connected or log on to the operating system from your terminal? The first operating system command that you will learn is the *logon command*, which brings your user identification code to the attention of the operating system. The logon procedure varies from one computer installation to another, so you must ask how to log on.

```
Logon Instructions
                  _____________________________________________
                  _____________________________________________
                  _____________________________________________
                  _____________________________________________
                  _____________________________________________
```

How do you get access to the SAS System? Similarly, the command you enter to request SAS access is different for different operating systems, execution modes, and installations. To see a sample SAS access request, you can refer to the "Operating System Notes" appendix for your operating system and execution mode. However, you must find out exactly which command to use from your SAS Software Consultant.

```
SAS System Access
   SAS Batch Mode _____________________________________________
   SAS Noninteractive Mode_____________________________________
   SAS Line Mode_______________________________________________
   SAS Display Manager_________________________________________
```

Which terminal keys do you use to enter operating system commands and to prepare your SAS program? In addition to the standard alphabet and number keys, terminals come from the factory with certain built-in functions permanently assigned to labeled keys. Some examples on your terminal may be cursor movements, ENTER, RETURN, DELETE, and SHIFT or CAPS LOCK. The functions included in the standard keys preset by terminal hardware vary from one terminal to another.

Some terminals are also equipped with "extra" keys that are not given specific duties when the terminal is manufactured. Software—including the operating system, the text editor you located for preparing SAS programs, and SAS full-screen software itself—may assign certain commands to these available keys, referred to as *function keys*.

Thus, the same terminal key may have different functions depending on whether you are typing in an operating system command, working with a text editor, or using SAS full-screen software like the SAS display manager, as shown in **Table 1.1**.

Table 1.1 Control of Terminal Key Functions

ACTIVITY	KEY DEFINITIONS DEPEND ON		
	Operating system	Text editor	SAS full-screen editor
Logon command	X		
Text editor access	X		
File preparation		X	
SAS System access	X		
SAS execution modes			
batch		X	
noninteractive		X	
line mode	X		
display manager*			X
SAS/FSP software*			X
SAS/AF software*			X
SAS/IML software	X		

*Use the KEYS command for a list of key definitions for the active screen.

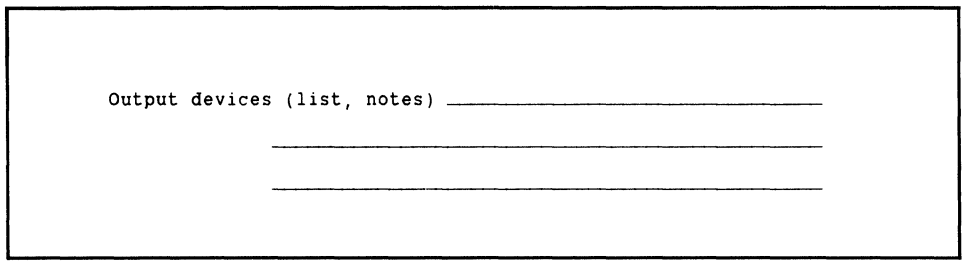
To answer the question that began this section, your current task determines the way you use your terminal keys from the time you log on to the computer until you receive your output. **Learning to use SAS software most effectively requires that you master your terminal's keyboard with operating system commands and with a text editor, if you need one, before you start.**

SEEING RESULTS

How will you receive the results of your SAS programs? As you become an experienced SAS user, you will probably use several different methods for getting SAS output for different applications.

Depending on execution mode, you receive the results of SAS programs on the terminal screen, in printed form, or in a computer file by default. Your computer facility may also have a local on-line access method your consultant can tell you about. If you are using SAS display manager with a full-screen terminal, you will be able to view the results of each program statement and any procedure output on special screens as soon as it is produced.

When you are ready to print the results, ask where output is printed by default. In addition, find out the control statements required to print the output on other devices in other locations, for example, a line printer in your department or a laser printer with high-quality results in another building. See the "SAS Log and Procedure Output" and the "SAS Display Manager" chapters for other considerations.



CALLING FOR HELP

Read the next chapter, "Introduction to the SAS Language," for an overview and a simple example. For more information from an interactive SAS session, use the SAS HELP facility. Enter a HELP command from any SAS screen or make HELP the next statement in your SAS program.

Chapter 2

Introduction to the SAS® Language

Like any language, the SAS language has its own vocabulary and syntax—words and the rules for putting them together. You define your data and the questions you have about them using the SAS language, and this sequence of SAS statements is called a *SAS program*.

SAS STATEMENTS

A *SAS statement* is a string of SAS keywords, SAS names, and special characters and operators ending in a semicolon that requests SAS to perform an operation or gives SAS information.

Here are some examples of SAS statements:

```
PUT X $15.;
```

```
DATA ONE;

FORMAT VALUE1 ABCD.;

PROC MEANS DATA=STORE.SUPPLY MAXDEC=3;

INFILE RAWDATA;

DO I=1 TO DIM(EACHITEM);

VAR %MACODE;

KEY1: TOTAL + 1;
```

SAS Keywords

Most SAS statements begin with a keyword that identifies the kind of statement it is (special cases are assignment, sum, comment, and null statements). The SAS statements shown above do not form a SAS program but are single SAS statements referred to by their keywords as a PUT statement, a DATA statement, a FORMAT statement, a PROC statement, an INFILE statement, an iterative DO statement, and a VAR statement. The last statement, preceded by the statement label KEY1, is a sum statement that does not have a keyword.

SAS Names

Among the kinds of names that can appear in SAS statements are the names of variables, SAS data sets, formats, procedures, options, arrays, statement labels, macros, as well as librefs and filerefs, which are special SAS references to files.

SAS names can be up to 8 characters long. The first character must be a letter (A,B,C,...,Z) or underscore (_). Later characters can be letters, numbers (0,1,...,9), or underscores.

Blanks cannot appear in SAS names, and special characters (for example, $,@,#), except for the underscore, are not allowed. The system reserves certain names that both begin and end with underscores for special variables (for example, _N_, _ERROR_, _I_), words that begin with a percent sign (%) for macro keywords, and words that begin with an ampersand (&) for references to macro variables.[1]

In the sample SAS statements shown earlier,

- X, VALUE1, and TOTAL are variable names.
- ONE and STORE.SUPPLY are SAS data set names; STORE is a libref.
- the items $15. and ABCD. are format names.
- MEANS is a procedure name.
- DATA= and MAXDEC= are options in the MEANS statement.
- RAWDATA is a fileref.
- in the DO statement, DIM() is a function name, and EACHITEM is the name of an array.
- KEY1 is a statement label.
- %MACODE is a macro call.
- TOTAL + 1 is an expression, which includes the constant 1.

Special Characters and Operators

Every SAS statement ends with a semicolon (;). Other special characters and operators illustrated in the statements above are parentheses (), the dollar sign $,

period ., equal sign =, colon :, addition operator or plus sign +, and the percent sign %.

Statement Descriptions

In this manual the form of a SAS statement is specified with these conventions:

KEYWORD *parameter...*[*item* | *item* | *item*] *options*;

where

bold
: indicates that you use exactly the same spelling and form as shown.

italics
: mean that you supply your own information.

[bracketed information]
: is optional. (However, you enter brackets shown in bold as part of the statement.)

parameters not in brackets
: are not optional.

...
: means that more than one of the parameters preceding ... can be optionally specified.

vertical bar (|) separating keywords and options
: means to choose this | or this.

options
: are keyword options specific to a particular SAS statement.

For the syntax of SAS statements, see the SAS Reference Card at the back of this manual.

A SAS PROGRAM

The statements in a SAS program are divided into two kinds of steps: DATA steps and PROC steps, the building blocks of all SAS programs. Usually DATA steps create SAS data sets, and PROC steps process SAS data sets, which are special SAS files for organizing and storing data. A SAS program is made up of either a DATA step or a PROC step, or both DATA and PROC steps. DATA and PROC steps can appear in any order, and any number of DATA or PROC steps can be used in a SAS program.

DATA Steps

The DATA step can include statements asking SAS to create one or more new SAS data sets and programming statements that perform the manipulations necessary to build the data sets. The DATA step begins with a DATA statement and can include any number of program statements. Report writing, file management, and information retrieval are all handled in DATA steps.

PROC Steps

The PROC step (or PROCEDURE step) asks SAS to call a procedure from its library and to execute that procedure, usually with a SAS data set as input. The PROC step begins with a PROC statement. Other statements in the PROC step give the program more information about the results that you want. The statements that

are available for your use in each PROC step depend on the specific SAS procedure that is called. Thus, each procedure description gives the statements that may accompany the PROC statement for that procedure.

SAS DATA SETS

A *SAS data set* is a collection of data values arranged in the rectangular form shown in **Table 2.1**.[2] The number of SAS data sets that you can use in a SAS job is limited only by space requirements.

Table 2.1 SAS Data Set Listing of Data Values

HODGES	191	36	50	5	162	60
KERR	189	37	52	2	110	60
PUTNAM	193	38	58	12	101	101
ROBERTS	162	35	62	12	105	37
BLAKE	189	35	46	13	155	58
ALLEN	182	36	56	4	101	42
HOWARD	211	38	56	8	101	38
VINCENT	167	34	60	6	125	40
STEWART	176	31	74	15	200	40
PERRY	154	33	56	17	251	250
HOWELL	169	34	50	17	120	38
ELLIS	166	33	52	13	210	115
SMITH	154	34	64	14	215	105
CROWE	147	46	50	1	50	50
CARTER	193	36	46	6	70	31
MOORE	202	37	62	12	210	120
LEE	176	37	54	4	60	25
VARNER	157	32	52	11	230	80
STONE	156	33	54	15	225	73
SCOTT	138	33	68	2	110	43

DATA VALUES

The basic unit that SAS works with is the *data value*.

Consider the following example. A researcher collects data from men in an exercise program. Each man's weight, pulse, and waist measurements are recorded, as well as the number of chin-ups, sit-ups, and jumps he can do before tiring.

Each of the measurements—the first man's weight, the second man's pulse, the last man's chin-ups—is a data value.

OBSERVATIONS

The data values associated with a single entity—an individual, a record, a year, a geographic region, an experimental animal—make up an *observation*. In **Table 2.1**, each row represents one observation. The first observation represents all the

data values associated with the first man whose measurements were recorded. The last observation represents all the data values for the last man.

The only limit to the number of observations that a SAS data set can contain is the space available on disk or tape to store the observations.

VARIABLES

The set of data values that describe a given characteristic make up a *variable*. Each observation in a SAS data set contains one data value for each variable. In **Table 2.1** each column of data values is a variable. For example, the first column makes up the variable NAME and contains all the names of the men in the club, the second column makes up the variable WEIGHT and contains their weight measurements, and so on.

The maximum number of variables in a SAS data set is limited only by the maximum size of an individual observation (see "SAS Files").

Variable Attributes

SAS variables are of two types: numeric and character. In addition to their type, SAS variables have these attributes: length, informat, format, and label. Variable attributes are either explicitly specified or defined from the context at their first occurrence.

Type Values of a numeric variable can only be numbers. In the fitness example all of the exercise values recorded are numeric; their values are measurements represented by numbers. Numeric values in the data lines can be preceded by plus or minus signs, decimal points can be included, but commas cannnot appear. The range of acceptable numeric values is determined by the range of your computer.[3]

Data values can also be character; letters and special characters as well as numeric digits can make up character data values. In the fitness data, the names of the men in the NAME variable are character data values. The dollar sign ($) following the name of a variable in some SAS statements used in the DATA step indicates that the data values are character rather than numeric. Character data values in SAS can range from 1 to 200 characters long.

Length The length attribute of a variable is the number of bytes used to store each of its values in a SAS data set. The default length is 8. (To store a variable in a length different from the default and for a discussion of other length considerations, see the **LENGTH Statement**.)

Informat A variable's informat is the pattern that SAS uses to read data values into the variable. The default informat is *w*. for numeric variables, $*w*. for character variables. (To use an informat other than the default, see the **INPUT Statement** and the **INFORMAT Statement**. The *w*. informat and other SAS informats are described in "SAS Informats and Formats.")

Format A variable's format is the pattern SAS uses to write each value of a variable. The default format is *w*. for numeric variables, $*w*. for character variables. (To use a format other than the default, see the **FORMAT Statement**. The *w*. and $*w*. formats and other SAS formats are described in "SAS Informats and Formats." To define your own formats, see "The FORMAT Procedure.")

Label The label attribute of a variable is a descriptive label of up to 40 characters that can be printed by certain procedures instead of the variable name. The

default label is blank. (For information on how to specify a variable label, see the **LABEL Statement**.)

Table 2.2 SAS Variable Attributes

VARIABLE ATTRIBUTE	POSSIBLE VALUES	DEFAULT VALUE*	ATTRIBUTE CAN BE SPECIFIED IN...
Type	numeric character	numeric	LENGTH statement**
Length numeric character	 2 to 8 bytes 1 to 200 bytes	 8 bytes 8 bytes	 LENGTH statement LENGTH statement
Informat numeric character	(see Chapter 15)	 *w.* *$w.*	 INFORMAT statement INFORMAT statement
Format numeric character	(see Chapter 15)	 *w.* *$w.*	 FORMAT statement FORMAT statement
Label	up to 40 characters	blank	LABEL statement

PUTTING TOGETHER DATA AND PROC STEPS

The DATA step that describes the collection of fitness data to SAS looks like this:

```
DATA FITNESS;
   INPUT NAME $ WEIGHT WAIST PULSE CHINS SITUPS JUMPS;
   CARDS;
HODGES   191  36  50   5  162   60
KERR     189  37  52   2  110   60
more data lines
;
```

- The DATA statement tells SAS to create a SAS data set named FITNESS.
- The INPUT statement tells SAS to read the data and gives the order and type of the data values.
- The CARDS statement tells SAS that the lines containing the data values follow immediately.
- The null statement (;) signals the end of the data lines.

This PROC step asks SAS to sort the observations according to the NAME variable

```
PROC SORT;
   BY NAME;
```

Since no other data set is specified, SAS uses the most recently created data set, which is the FITNESS data set.

The PROC statement that asks SAS to print a list of the data values looks like this:

```
PROC PRINT;
   TITLE 'FITNESS DATA';
```

These statements tell SAS to use the PRINT procedure to list the data values titled FITNESS DATA.

The PROC step that asks SAS to find the mean values for the fitness measurements is

```
PROC MEANS MAXDEC=1;
```

This statement tells SAS to execute the MEANS procedure to get means and other summary statistics for all the numeric variables in the SAS data set. The MAXDEC=1 option specifies that the statistics should be printed with one decimal place.

These PROC step statements

```
PROC CORR;
   VAR WEIGHT WAIST PULSE CHINS;
```

ask for the CORR procedure and use a VAR statement to request correlations for the four variables WEIGHT, WAIST, PULSE, and CHINS.

WRITING SAS PROGRAMS

The SAS language allows flexibility in the way SAS statements are written in a program.

Required Spacing

SAS statements can begin in any column of a line, and several statements can be written on the same line. You can begin a statement on one line and continue it to another line as long as no word is split.

You need at least one blank between each separate item in a SAS statement as in the statements above. Some special characters, such as the equal sign after a word, can take the place of a blank, although blanks are always allowed.

For example, in the statement

```
TOTAL2=TOTAL+10;
```

you do not need a blank before or after the equal sign or before or after the plus sign because the equal sign and the plus sign are special characters, but you can also have any number of extra blanks. The statement

```
TOTAL2  =  TOTAL + 10 ;
```

is equivalent to the statement above.

(Although SAS does not have rigid spacing requirements, SAS programs are easier to read if the statements are indented consistently. The examples in this manual illustrate our spacing conventions.)

Comments

Comments of the form /* COMMENTS HERE */ can appear within SAS statements wherever a single blank can appear. For example, the statement

```
PROC SORT /* SORT THE DATA SET */;
```

is a valid SAS statement.

Variable Lists

Variables are defined in the order in which they first appear. In the INPUT statement above, NAME is the first variable defined, followed by WEIGHT, WAIST, PULSE, CHINS, SITUPS, and JUMPS. In a DATA step with a SET, MERGE, or UPDATE statement or in a PROC step, the currently defined names are the names of the variables in the data set being processed.

After a complete list of variables has been defined in a SAS program, you can use abbreviated variable lists in many later SAS statements.

The different kinds of abbreviated lists are shown in **Table 2.3**.

Table 2.3 SAS Variable Lists

NAMES OF THE FORM...	CAN BE ABBREVIATED...	TO REPRESENT...
Numbered names of the form X1,X2,...,X*n*	X1—X*n*	all variables from X1 to X*n*
Ranges of names of the form X P A*	X—A	all variables from X to A
	X-NUMERIC-A	all numeric variables from X to A
	X-CHARACTER-A	all character variables from X to A
Special SAS names	_NUMERIC_	all numeric variables
	CHARACTER	all character variables
	ALL	all variables

For example, the INPUT statement

```
INPUT NAME $ WEIGHT WAIST PULSE CHINS SITUPS JUMPS;
```

can also be written to include a numbered variable list:

```
INPUT NAME $ VAR1-VAR6;
```

Note that the character variable NAME is not included in this abbreviated list. Variables in a numbered list need not be but are usually of the same type, numeric or character.

The VAR statement for the CORR procedure

```
VAR WEIGHT WAIST PULSE CHINS;
```

in the example can also be abbreviated as a range:

```
VAR WEIGHT--CHINS;
```

Other examples of abbreviated variable lists are shown throughout this manual. (Note: abbreviated variable lists are not allowed in the SAS data set options described in "SAS Files.")

OUTPUT FROM A SAS PROGRAM

Below is the complete SAS program for the fitness data including all the data lines and the output it produces. For a detailed discussion of SAS output features, see "SAS Log and Procedure Output."

```
DATA FITNESS;
   INPUT NAME $ WEIGHT WAIST PULSE CHINS SITUPS JUMPS;
   CARDS;
HODGES    191  36  50   5  162   60
KERR      189  37  52   2  110   60
PUTNAM    193  38  58  12  101  101
ROBERTS   162  35  62  12  105   37
BLAKE     189  35  46  13  155   58
ALLEN     182  36  56   4  101   42
HOWARD    211  38  56   8  101   38
VINCENT   167  34  60   6  125   40
STEWART   176  31  74  15  200   40
PERRY     154  33  56  17  251  250
HOWELL    169  34  50  17  120   38
ELLIS     166  33  52  13  210  115
SMITH     154  34  64  14  215  105
CROWE     147  46  50   1   50   50
CARTER    193  36  46   6   70   31
MOORE     202  37  62  12  210  120
LEE       176  37  54   4   60   25
VARNER    157  32  52  11  230   80
STONE     156  33  54  15  225   73
SCOTT     138  33  68   2  110   43
;
PROC SORT;
   BY NAME;
PROC PRINT;
   TITLE 'FITNESS DATA';
PROC MEANS MAXDEC=1;
PROC CORR;
   VAR WEIGHT WAIST PULSE CHINS;
```

Output 2.1 SAS Log from Fitness Job

```
1          SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB INTRLANG STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB INTRLANG HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

1          DATA FITNESS;
2             INPUT NAME $ WEIGHT WAIST PULSE CHINS SITUPS JUMPS;
3             CARDS;

NOTE: DATA SET WORK.FITNESS HAS 20 OBSERVATIONS AND 7 VARIABLES. 317 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.96 SECONDS AND 376K.

24         ;
25         PROC SORT;
26            BY NAME;
```

(continued on next page)

(continued from previous page)

```
NOTE: 4 CYLINDERS DYNAMICALLY ALLOCATED ON RIO FOR EACH OF 3 SORT WORK DATA SETS.
NOTE: DATA SET WORK.FITNESS HAS 20 OBSERVATIONS AND 7 VARIABLES. 317 OBS/TRK.
NOTE: THE PROCEDURE SORT USED 1.08 SECONDS AND 668K.

27         PROC PRINT;
28            TITLE 'FITNESS DATA';
NOTE: THE PROCEDURE PRINT USED 0.84 SECONDS AND 480K AND PRINTED PAGE 1.

29         PROC MEANS MAXDEC=1;
NOTE: THE PROCEDURE MEANS USED 0.85 SECONDS AND 500K AND PRINTED PAGE 2.

30         PROC CORR;
31            VAR WEIGHT WAIST PULSE CHINS;
NOTE: THE PROCEDURE CORR USED 0.84 SECONDS AND 492K AND PRINTED PAGE 3.
NOTE: SAS USED 668K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Output 2.2 PROC PRINT Listing of FITNESS Data Set

FITNESS DATA 1

OBS	NAME	WEIGHT	WAIST	PULSE	CHINS	SITUPS	JUMPS
1	ALLEN	182	36	56	4	101	42
2	BLAKE	189	35	46	13	155	58
3	CARTER	193	36	46	6	70	31
4	CROWE	147	46	50	1	50	50
5	ELLIS	166	33	52	13	210	115
6	HODGES	191	36	50	5	162	60
7	HOWARD	211	38	56	8	101	38
8	HOWELL	169	34	50	17	120	38
9	KERR	189	37	52	2	110	60
10	LEE	176	37	54	4	60	25
11	MOORE	202	37	62	12	210	120
12	PERRY	154	33	56	17	251	250
13	PUTNAM	193	38	58	12	101	101
14	ROBERTS	162	35	62	12	105	37
15	SCOTT	138	33	68	2	110	43
16	SMITH	154	34	64	14	215	105
17	STEWART	176	31	74	15	200	40
18	STONE	156	33	54	15	225	73
19	VARNER	157	32	52	11	230	80
20	VINCENT	167	34	60	6	125	40

Output 2.3 PROC MEANS Results for Fitness Data

FITNESS DATA 2

VARIABLE	N	MEAN	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	STD ERROR OF MEAN	SUM	VARIANCE	C.V.
WEIGHT	20	173.6	19.7	138.0	211.0	4.4	3472.0	389.6	11.4
WAIST	20	35.4	3.2	31.0	46.0	0.7	708.0	10.3	9.0
PULSE	20	56.1	7.2	46.0	74.0	1.6	1122.0	52.0	12.9
CHINS	20	9.4	5.3	1.0	17.0	1.2	189.0	27.9	55.9
SITUPS	20	145.5	62.6	50.0	251.0	14.0	2911.0	3914.6	43.0
JUMPS	20	70.3	51.3	25.0	250.0	11.5	1406.0	2629.4	72.9

Output 2.4 PROC CORR Results for Fitness Data

```
                                    FITNESS DATA                                3

VARIABLE       N            MEAN         STD DEV            SUM         MINIMUM         MAXIMUM

WEIGHT        20     173.60000000     19.73882095     3472.00000000     138.00000000     211.00000000
WAIST         20      35.40000000      3.20197308      708.00000000      31.00000000      46.00000000
PULSE         20      56.10000000      7.21037265     1122.00000000      46.00000000      74.00000000
CHINS         20       9.45000000      5.28627817      189.00000000       1.00000000      17.00000000

          PEARSON CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 20

                                WEIGHT     WAIST     PULSE     CHINS

                    WEIGHT     1.00000   0.20585  -0.23194  -0.06123
                                0.0000    0.3839    0.3251    0.7976

                    WAIST      0.20585   1.00000  -0.35289  -0.55223
                                0.3839    0.0000    0.1270    0.0116

                    PULSE     -0.23194  -0.35289   1.00000   0.15065
                                0.3251    0.1270    0.0000    0.5261

                    CHINS     -0.06123  -0.55223   0.15065   1.00000
                                0.7976    0.0116    0.5261    0.0000
```

NOTES

1. Other examples of automatic special variables used by SAS under certain circumstances include _COL_, _FDBK_, _FREQ_, _IORC_, _LABEL_, _LNDET_, _MODEL_, _NAME_, _PRIOR_, _RBA_, _ROW_, _RRN_, _SIGMA_, _TITLES_, _TYPE_, and _WEIGHT_.

2. The data for the example in this chapter were provided by A.C. Linnerud, North Carolina State University.

3. **AOS/VS and PRIMOS**: range is 5.4E−79 to 7.2E+75.
   VMS: range is 2.9E−39 to 1.7E+38.
   CMS, OS, VM/PC, and VSE: range is 10E-73 to 10E+73.

4. **PRIMOS**: acceptable length of a numeric variable is 3 to 8 bytes.

5. **AOS/VS, PRIMOS, and VMS**: suffixes must be the same length if either suffix contains leading zeros. For example, X1−X100 is acceptable but X01-X100 is not acceptable.

THE DATA STEP

Introduction to the DATA Step

SAS® Statements Used in the DATA Step

SAS® Expressions

SAS® Functions

DATA Step Applications

Chapter 3

Introduction to the DATA Step

Introduction

Before you can use SAS software to prepare your data for analysis or use a SAS procedure to analyze your data, you must first get them into a SAS data set. Once your data are in a SAS data set, you can combine the data set with other SAS data sets in many different ways and use any of the SAS procedures.

The DATA step includes statements asking SAS to create one or more new data sets and programming statements that perform the manipulations necessary to build the data sets. Report writing, file management, and information retrieval are all handled in DATA steps.

You can use the DATA step for these purposes:

- retrieval—getting your input data into a SAS data set
- editing—checking for errors in your data and correcting them; computing new variables
- printing reports according to your specifications and writing disk or tape files

- producing new SAS data sets from existing ones by subsetting, merging, and updating the old data sets.

What Is a DATA Step?

A DATA step is a group of SAS statements that begins with a DATA statement and usually includes all the statements in one of these three groups:

Data on disk or tape

```
DATA statement;
  INFILE statement;
  INPUT statement;
  other SAS statements used in the DATA step
```

Data in job stream

```
DATA statement;
  INPUT statement;
  other SAS statements used in the DATA step
  CARDS statement;
data lines
;
```

Data in existing SAS data set

```
DATA statement;
  SET | MERGE | UPDATE statement;
  other SAS statements used in the DATA step
```

In the last DATA step the bar (|) notation indicates that you can use either a SET, MERGE, or UPDATE statement.

A DATA step can also include any of the other statements described in "Statements Used in the DATA Step" and briefly introduced later in this chapter.

Creating SAS Data Sets

The three types of simple DATA steps and a special DATA step for writing reports are shown below.

Data on disk or tape The form of a DATA step for producing SAS data sets from input data on disk or tape is

```
DATA statement;
  INFILE statement;
  INPUT statement;
  other SAS statements used in the DATA step
```

The statements in the DATA step above have these functions:

- The DATA statement signals the beginning of the DATA step and gives a name to the SAS data set you are creating.
- The INFILE statement gives the fileref of the control statement describing the file that contains the data. When the INFILE statement is executed, the external file is opened. (See the "Operating System Notes" appendix for a definition of a fileref for your operating system.)
- The INPUT statement describes your input data, giving a name to each variable and identifying its location on the disk or tape file.
- If you are modifying the data, optional program statements give SAS directions for the changes you want.

In-stream data The form of a DATA step for producing a SAS data set from input in the job stream with the SAS statements is

```
DATA statement;
  INPUT statement;
  other SAS statements used in the DATA step
  CARDS statement;
data lines
;
```

If you enter data from a terminal or if your data are on cards, you still need to describe the data's format to SAS in an INPUT statement. The required statements are identical to those in the example above, except that

- no INFILE statement is needed
- a CARDS statement immediately precedes the data lines, signaling the end of the statements and the beginning of the data for the DATA step.

Data from other SAS data sets The form of a DATA step for producing a SAS data set from one or more existing data sets is

```
DATA statement;
  SET | MERGE | UPDATE statement;
  other SAS statements used in the DATA step
```

You can create a SAS data set from one or more existing SAS data sets. For example, if you have questionnaire data stored in a SAS data set, you may want to build another data set containing only the responses for males. Rather than read the original raw data again to produce a data set for males only, it is easier to use a SET statement to read the SAS data set, selecting only those observations where the SEX value is M.

- The DATA statement tells SAS to start this DATA step and name the new data set.
- The SET, MERGE, or UPDATE statement identifies the old SAS data sets. (Optionally, a BY statement gives the identifying variables for the SET, MERGE or UPDATE.)
- Optional program statements give SAS directions for modifying the data.

Writing reports The form of a DATA step for producing a report is

```
DATA _NULL_;
  INPUT and (CARDS | INFILE) statement;
    or
  SET | MERGE | UPDATE statement;
  FILE statement;
  PUT statement;
  other SAS statements used in the DATA step
```

If you want to report your data values in a form different from that produced by SAS reporting procedures, you can do it in a DATA step:

- The DATA _NULL_ statement tells SAS to begin the DATA step; using the special name _NULL_ means that a SAS data set is not produced.
- To provide the input data for the report, you need either INPUT and CARDS or INFILE statements with accompanying data; or SET, MERGE, or UPDATE statements; or program statements to generate data.
- The FILE statement tells SAS where to print the report or write the file.
- Optionally, you can use program statements to compute some new values.

- One or more PUT statements write the lines of the report or file.

DATA Step Statements

The SAS statements that can appear in a DATA step fall into several categories: file-handling statements, action statements, control statements, and information statements. Each statement is also either executable, positional, or declarative.

- Executable statements (denoted by an X in the discussion below) are programming statements that cause some action.
- Positional statements (P) cause no action at execution, but their position in the DATA step is important.
- Declarative statements (D) supply additional information to SAS. Their position in the step is not usually important.

Other statements available for use within a DATA step are described in "SAS Statements Used Anywhere."

File-handling statements File-handling statements let you work with files used as input to the data set or files to be written by the DATA step. SAS's file-handling statements are

CARDS precedes card data or lines entered at a terminal—data that are part of the job stream. (P)

Operating systems: All

CARDS4 precedes in-stream data lines containing semicolons. (P)

Operating systems: All

DATA tells SAS to begin a DATA step and to start building a SAS data set. (P)

Operating systems: All

FILE identifies the external file where lines are to be written by the DATA step. (X)

Operating systems: All

INFILE identifies the external file containing raw input data to be read by the DATA step. (X)

Operating systems: All

INPUT describes the records on the external input file. (X)

Operating systems: All

MERGE combines observations from two or more SAS data sets into a new data set. (X)

Operating systems: All

PUT describes the format of the lines to be written by SAS. (X)

Operating systems: All

SET reads observations from one or more existing SAS data sets. (X)

Operating systems: All

UPDATE applies transactions to a master file. Both transaction and master file are SAS data sets. (X)

Operating systems: All

Action statements While creating a SAS data set in a DATA step, you may want to modify your data from the way they appear on the input lines, select only certain observations for the data set being created, or look for errors in the input data. Action statements allow you to work with observations as they are being created. SAS action statements are

ABORT stops the current DATA step or the job, depending on the mode of executing SAS you are using. (X)

Operating systems: All

assignment creates and modifies variables. (X)

Operating systems: All

CALL invokes or calls a routine. (X)

Operating systems: All

DELETE excludes observations from the data set being created. (X)

Operating systems: All

ERROR writes messages on the SAS log. (X)

Operating systems: CMS, OS, VM/PC, and VSE

LIST lists input lines. (X)

Operating systems: All

LOSTCARD corrects for lost data lines when an observation has an incorrect number of data lines. (X)

Operating systems: All

MISSING declares that certain values in the input data represent special missing values for numeric data. (D)

Operating systems: All

null holds a place for a label or signals the end of data lines. (P)

Operating systems: All

OUTPUT creates new observations. (X)

Operating systems: All

STOP stops creating the current data set. (X)

Operating systems: All

subsetting IF selects observations for the data set being created. (X)

Operating systems: All

sum accumulates totals. (X)

Operating systems: All

Control statements SAS statements in a DATA step are executed one by one for each observation. In some cases, you may want to skip statements for certain observations or to change the order of the statements encountered. SAS statements that let you transfer control from one part of the program to another are called *control statements*. SAS control statements are

DO, iterative DO, DO OVER, DO UNTIL, DO WHILE — sets up a group of statements to be executed as one statement (DO) or iteratively (iterative DO); for each element in an implicitly subscripted array (DO OVER); until some condition is true (DO UNTIL); or until some condition is no longer true (DO WHILE). (X)

Operating systems (DO, iterative DO, DO UNTIL, and DO WHILE): All

Operating systems (DO OVER): CMS, OS, VM/PC, and VSE

END — signals the end of a DO group. (P)

Operating systems: All

GO TO — causes SAS to jump to a labeled statement in the step and continue execution at that point. (X)

Operating systems: All

IF-THEN/ELSE — conditionally executes a SAS statement. (X)

Operating systems: All

LINK-RETURN — causes SAS to jump to a labeled statement in the step and execute statements until it encounters a RETURN statement. (X)

Operating systems: All

RETURN — when not combined with a LINK statement, causes SAS to return to the beginning of the DATA step to begin execution. When combined with a LINK statement, returns to the statement immediately following the most recently executed LINK. (X)

Operating systems: All

SELECT — conditionally executes one of several SAS statements. (X)

Operating systems: All

Information statements In the last category of SAS statements are information statements. These statements give SAS extra information about the data set or sets being created. Information statements are not executable and can appear anywhere in the DATA step with the same effect. The information provided in the statement takes effect either prior to the execution of the step or at the time observations are written to the SAS data set. SAS information statements are

ARRAY — defines a set of variables to be processed the same way. (D)

Operating systems: All

ATTRIB — specifies a format, informat, label, and length for a variable. (D)

Operating systems: All

BY specifies that the data set is to be processed in groups defined by the BY variables. (D)

Operating systems: All

DROP identifies variables to be excluded from a data set or analysis. (D)

Operating systems: All

FORMAT specifies formats for printing variable values. (D)

Operating systems: All

INFORMAT specifies informats for storing variable values. (D)

Operating systems: All

KEEP identifies variables to be included in a data set or analysis. (D)

Operating systems: All

LABEL associates descriptive labels with variable names. (D)

Operating systems: All

LENGTH specifies the number of bytes to be used for storing SAS variables. (D)

Operating systems: All

RENAME changes the names of the variables in a data set. (D)

Operating systems: All

RETAIN identifies variables whose values are not to be set to missing each time the DATA step is executed and can give variables an initial value. (D)

Operating systems: All

DATA Step Flow

The DATA statement that begins each DATA step always signals the beginning of the step. The remaining statements can be called the "program" because SAS translates them to the computer's machine language and executes them each time it goes through the DATA step—usually, once for each observation in your input data.

In **Figure 3.1** you can see the normal flow of the DATA step for creating a new data set.

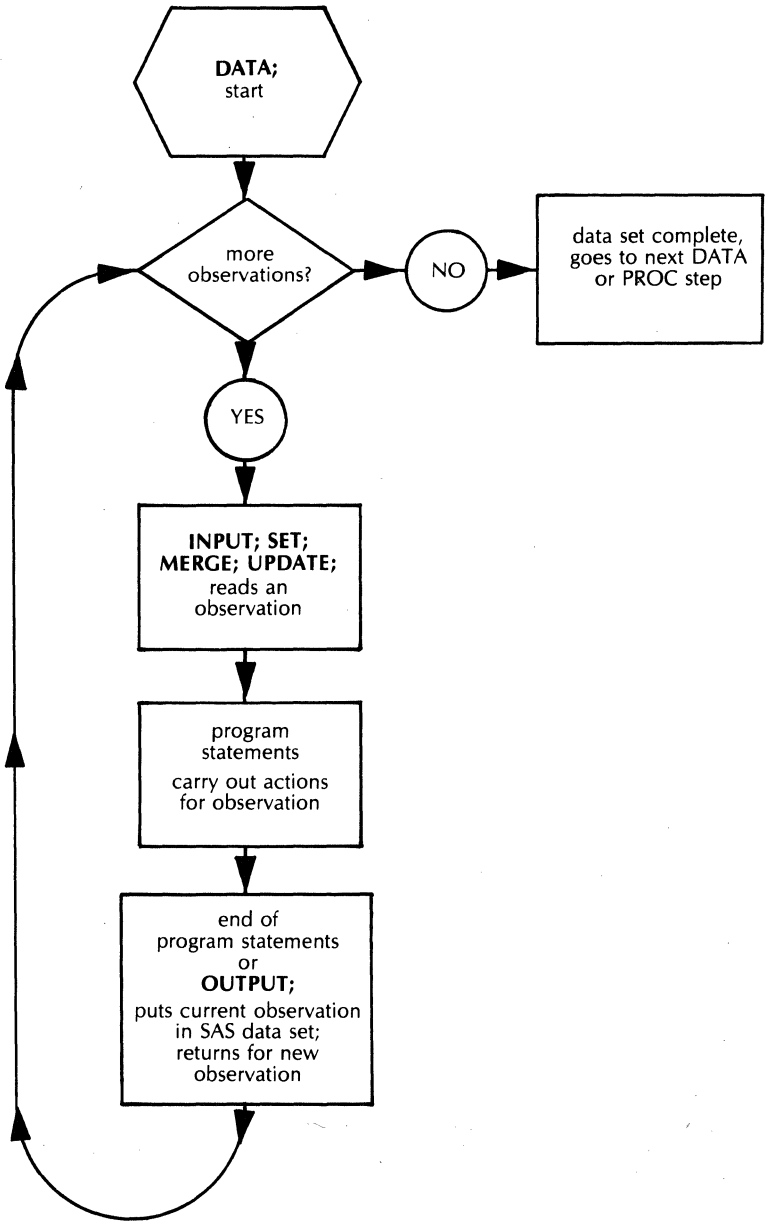


Figure 3.1 Flow of Action within DATA Step

SAS compiles the program into machine code. (SAS is a compiler; however, the machine code generated cannot be stored and reused without recompiling the SAS statements.) All the variables mentioned in the DATA step become part of the vector of current values, also called the *program data vector*. The variables from each source of input data, together with variables created by program statements, are in the program data vector and are available to program statements in the DATA step.

The new SAS data set or data sets can contain all of these variables, or you can choose any subset of variables to be output to the data set.

Variables read with an INPUT statement or created in programming statements are set to missing before each execution of the DATA step. Variables read with a SET, MERGE, or UPDATE statement are retained. SAS executes each statement in the step, building observations for the SAS data set.

When SAS executes the last statement in the DATA step (or a RETURN statement that causes a return to the beginning of the step for a new execution), it normally writes the current values from the program data vector to the SAS data set being created.

SAS returns to the first statement after the DATA statement, initializes nonretained variables in the program data vector to missing, and begins executing statements to build the next observation.

When SAS has read and processed all the data from any of the input files, it goes on to the next DATA or PROC step.

How many times is a DATA step executed? SAS goes through the statements in the DATA step for each record that it reads. So although each statement appears only once, SAS carries it out for every observation. A DATA step that does not contain an INPUT, SET, MERGE, or UPDATE statement is executed only once. Otherwise, the step is repeated until SAS runs out of data in one of the input sources or until a STOP or ABORT statement is executed. Program statements may be included that cause other statements in the DATA step to be executed many times: for example, DO loops, LINK/RETURN, or GO TO statements.

The SAS variable _N_ is automatically generated by SAS for each DATA step. Its value is the number of times SAS has begun executing the DATA step. You can use this variable in program statements in the DATA step. For example, to execute the statements in a DO group the first time through the DATA step, use the statement

```
IF _N_=1 THEN DO;
```

How SAS executes a DATA step: example 1 A typical SAS job includes an INPUT statement that describes data values on cards, disk, or tape. The variables given in this INPUT statement make up the program data vector. If any program statements create new variables, these new variables also become part of the program data vector. For example, consider this DATA step:

```
DATA FITNESS;
   INPUT WEIGHT WAIST JUMPS SITUPS PULSE;
   RATIO=PULSE/JUMPS;
   CARDS;
data lines
;
```

These statements ask SAS to create a data set named FITNESS, to read five variables for this data set from input data lines, and to create a sixth variable, RATIO. The program data vector for this DATA step contains six variables:

WEIGHT WAIST JUMPS SITUPS PULSE RATIO

SAS executes this DATA step once for each observation.

SAS reads the current record's data values using the directions in the INPUT statement. SAS carries out the program statement, adding to the program data vector the value of RATIO.

After the last program statement, the values in the program data vector (the observation) are automatically added to the data set FITNESS being created. (This

automatic outputting is equivalent to what would happen if an OUTPUT statement were present.)

SAS returns for another execution of the DATA step. (This automatic return is equivalent to what would happen if a RETURN statement were present.)

If there were twenty data lines, each containing the five input values, this DATA step would be executed twenty times; the new data set FITNESS would contain twenty observations. Each observation would contain six variables.

Data from two sources: example 2 It is possible for a DATA step to contain several INPUT, SET, MERGE, or UPDATE statements. In such cases, the observations contain variables contributed by each statement. For example, consider this DATA step:

```
DATA FITNESS;
   INPUT WEIGHT WAIST JUMPS SITUPS PULSE;
   SET MORE;
   RATIO=AGE/JUMPS;
   DROP AGE JUMPS HEIGHT;
   CARDS;
data lines
;
```

The program data vector for this DATA step is like that in the example above, except that it also includes whatever variables the SAS data set MORE contains. If MORE contains the variables AGE and HEIGHT, the program data vector contains the variables shown in **Figure 3.2.** Since a DROP statement appears, observations written to the new data set FITNESS contain only these variables:

WEIGHT WAIST SITUPS PULSE RATIO

Note that when a DATA step has more than one source of input data, the step ends when all the data have been read from one of the sources. Thus, the example above is normally used only if there are an equal number of raw data records and observations in data set MORE.

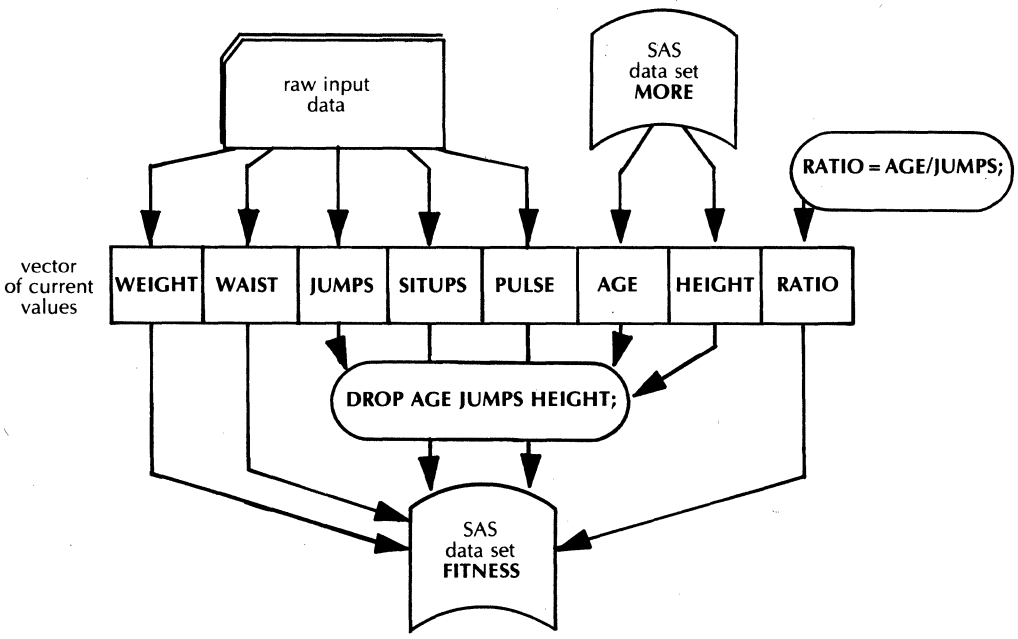


Figure 3.2 Program Data Vector: Data from Two Sources

Space Required for SAS Data Sets

This section gives formulas you can use to calculate the amount of space required for SAS data sets on disk under your operating system. You do **not** need to calculate space every time you create a SAS data set, but you can use these formulas to get an idea of the size of average SAS data sets you create.

In the following formulas,

- CEIL is a SAS function that returns the smallest integer greater than or equal to an argument.
- FLOOR is a SAS function that returns the largest integer less than or equal to an argument.
- NVAR is the number of variables.
- NOBS is the number of observations.

  If you have already run a DATA step for a data set, you can get the number of observations from the note that SAS prints on the log.
- LRECL is the length of the data in an observation.

  If the data set contains all numeric variables of default length, LRECL= 8*NVAR+4. When the data set contains character variables or variables of length other than 8, sum the lengths of all variables and add 4 to compute LRECL.

AOS/VS and VMS The space for SAS data sets is allocated automatically by the system within the disk quota limitations established for your user account. Use the following formula to determine how much disk space is required for a SAS data set prior to actually creating the data set:

```
SIZE=CEIL((260+NVAR*136+NOBS*LRECL)/512)
```

where SIZE is the number of disk units needed (AOS/VS: disk blocks; VMS: disk blocks for FILES-11 disk volumes).

CMS and VM/PC Each member of a SAS data library is a separate minidisk file. The following formula calculates the space required for each data set.

```
SIZE=CEIL((98+372+HISTLEN+(NVAR*82)+(NOBS*(LRECL+4)))/BLKSIZE);
```

where

- SIZE is the number of CMS minidisk blocks that the SAS data set needs.
- GEN is the number of generations of data set history and descriptor information stored with the data set.
- SRC is the number of source statements in the SAS DATA or PROC step that creates the data set.
- BLKSIZE is the CMS disk block size, as returned by the CMS command QUERY DISK.
- HISTLEN is the length of the history data stored with the data set. If the SAS data set option or global system option GEN=0 is in effect when the data set is created, HISTLEN will have the value zero. If the GEN option has a nonzero value, there will be between one and that many generations of history data stored on the data set at any given time.

  To compute the value of HISTLEN, use the following formula, expressed below as a series of SAS programming statements:

```
HISTLEN=(GEN-1)*372;
DO I=1 TO GEN BY 1;
   HISTLEN=HISTLEN+(SRC(I)*90)+(FILE(I)*184);
END;
```

  where

- I is the DO loop counter variable.
- GEN is the number of generations of history data actually stored with the data set. This is not necessarily the current value of the GEN= global or data set option. The HISTORY option for PROC CONTENTS can be used to determine the value of GEN.
- SRC is the number of source statements in the *i*th generation of history data. To determine each value of SRC, count the number of source statements printed by PROC CONTENTS for each generation.
- FILE is the total number of INFILE and FILE statements in the *i*th generation of history data. Count the number of FILE and INFILE statements in the source statements printed by PROC CONTENTS for each generation to determine each value of FILE.

To make sure there is enough minidisk space to store the SAS data set, make sure that the SIZE value you obtained from the formula above is less than the BLKS LEFT value given by the CMS command QUERY DISK. For information on the QUERY DISK command, consult the IBM manual *CMS Command and Macro Reference*.

OS and VSE Each SAS data set consists of three components: a directory entry, a descriptor portion, and a data portion. To compute the storage size requirements for a SAS data set, you must compute the requirements for each component and add them.

```
SIZE=DIRSIZE+DESCSIZE+DATASIZE
```

To calculate the space a library needs, sum the SIZE values for each data set in the library, and add one or more tracks for the library's directory.

In the following formulas **TRACKLEN** is the length of a track on the disk device being used. Note that it is dependent on the blocking factor for the component under consideration. Increasing the number of physical blocks written to a track (by decreasing the BLKSIZE= SAS option) reduces the amount of the track available for writing data since each block consumes a certain amount of overhead. For estimates of data set size, TRACKLEN can be assumed to be the track capacity; however, for maximum precision, TRACKLEN should be the size, in bytes, of the track available for data. It can be computed by

```
TRACKLEN=FLOOR(TRACKCAP/(BLKSIZE+OVERHEAD))*BLKSIZE
```

where

- BLKSIZE is the block size used for the library component under consideration (data set descriptor portion, data set data portion, or directory).
- TRACKCAP is the total capacity of a track in bytes for the disk device under consideration.
- OVERHEAD is the size in bytes of overhead (inter-block gap) for the disk device under consideration.

Both TRACKCAP and OVERHEAD can be obtained from the IBM manual *Data Management Macro Instructions*.

A track corresponds to a control interval on FBA disks.

It is very difficult to be precise about predicting the size requirements for the directory and descriptor portion of a particular SAS data set, so these formulas should be used with caution. It is possible to be fairly precise about predicting the size requirements for the data portion of a particular data set, which is usually the largest of the three components. Because the data portion may not begin on a track boundary, but instead may share a track with the descriptor portion, the

computed size may be slightly larger than required. In general, this is only true for small block sizes, such as for SAS data sets created under TSO (because a site may choose a smaller default for the BLKSIZE= option for foreground jobs than for background jobs).

To compute the size of the directory component, use the following formulas.

```
DIRNUM=CEIL(NEXTENTS/10)
DIRSIZE=CEIL(TRACKLEN/(DIRNUM*1024))
```

where

- DIRNUM is the number of directory records for each data set. Compute and sum DIRNUM for all data sets in the library before using it in the second formula.
- DIRSIZE is the number of tracks required for the directory records.
- NEXTENTS is the number of SAS data set extents as reported by PROC CONTENTS. NEXTENTS cannot be predicted with accuracy without a detailed knowledge of the library layout.

  Note that it is very difficult to predict the number of subextents a data set will use since it depends on the layout of existing data sets in the library. In a newly created library, a data set has one subextent, unless it was created simultaneously with one more data sets, such as with a DATA statement with more than one data set name.
- SAS library directories have a block size of 1024 bytes. Use this value to compute an accurate TRACKLEN for the formula.

To compute the size of the descriptor component, use the following formula.

```
DESCSIZE=CEIL((372+NVAR*82+HISTLEN)/TRACKLEN)
```

where

- DESCSIZE is the number of tracks needed for the descriptor portion of the data set.
- HISTLEN is the length of the history data stored with the data set. If the SAS data set option or global system option GEN=0 is in effect when the data set is created, HISTLEN will have the value zero. If the GEN option has a nonzero value, there will be between one and that many generations of history data stored on the data set at any given time.

  To compute the value of HISTLEN, use the following formula, expressed below as a series of SAS programming statements:

  ```
  HISTLEN=(GEN-1)*372;
  DO I=1 TO GEN BY 1;
     HISTLEN=HISTLEN+(SRC(I)*90)+(FILE(I)*184);
     END:
  ```

  where

  - I is the DO loop counter variable.
  - GEN is the number of generations of history data actually stored with the data set. This is not necessarily the current value of the GEN= global or data set option. The HISTORY option for PROC CONTENTS can be used to determine the value of GEN.
  - SRC is the number of source statements in the *i*th generation of history data. To determine each value of SRC, count the number of source statements printed by PROC CONTENTS for each generation.
  - FILE is the total number of INFILE and FILE statements in the *i*th generation of history data. Count the number of FILE and INFILE statements in the source statements printed by PROC CONTENTS for each generation to determine each value of FILE.

The descriptor portion of data sets has a block size of 1024 bytes. Use this value to compute an accurate TRACKLEN.

The formula for the size of the descriptor portion of a data set can only give an estimate since each generation of history information can be different. For example, create a SAS data set using several INFILE statements and several hundred lines of code. Then use that data set to create another using two or three lines of code. The size estimate will be too small. Conversely, reverse the situation and the estimate will be too large. (A liberal estimate of the space required to store data set history and descriptor information, other than header and variable descriptor records, is 3000 bytes.)

To compute the size of the data component, use the following formula.

```
DATASIZE=CEIL(NOBS/FLOOR(TRACKLEN/(LRECL+4)))
```

where DATASIZE is the number of tracks needed for the data portion of the data set.

PRIMOS The space for SAS data sets is allocated automatically by the system within the disk quota limitations established for your user account. Use the following formula to determine how much disk space is required for a SAS data set prior to actually creating the data set.

```
SIZE=CEIL((260+NVAR*136+NOBS*LRECL)/2048)
```

where SIZE is the number of disk records needed.

References

AOS/VS, PRIMOS, and VMS:

Technical Report U-104 "SAS® Data Set Organization Under the AOS/VS, PRIMOS, and VMS Operating Systems."

CMS and VM/PC:

CMS Command and Macro Reference, the QUERY command and the FORMAT command, International Business Machines Corporation Order Number SC19-6209.

The CMS User's Guide, Chapter 3: The CMS File System, International Business Machines Corporation Order Number SC19-6210.

OS and VSE:

OS Data Management Macro Instructions, Appendix C: Device Capacities, International Business Machines Corporation (1972) Order Number GC26-3794.

OS/VS1 Data Management Macro Instructions, 2nd ed., Appendix C: Device Capacities, International Business Machines Corporation (1983) Order Number GC26-3872.

OS/VS2 MVS Data Management Macro Instructions, Appendix C: Device Capacities, International Business Machines Corporation (1976) Order Number GC26-3873.

MVS/370 Data Management Macro Instructions, Appendix C: Device Capacities, International Business Machines Corporation (1983) Order Number GC26-4057.

MVS/XA Data Management Macro Instructions, Appendix C: Device Capacities, International Business Machines Corporation (1983) Order Number GC26-4014.

Chapter 4

SAS® Statements Used in the DATA Step

ABORT Statement

Operating systems: All

You can use the ABORT statement in a DATA step to cause the SAS System to cease executing the current DATA step immediately and go into syntax-check mode (in batch or noninteractive processing) or to cease executing the current DATA step and resume execution with the next DATA or PROC step (in display manager or interactive processing). The SAS System may take additional action depending on the ABORT statement options you specify. When the ABORT statement is executed, SAS prints a message and creates a data set containing the observations that were processed before the ABORT statement was executed. However, if the new data set has the same name as an existing SAS data set in the job or session, the existing data set is not replaced.

In batch and noninteractive processing, the SAS System also sets the value of the OBS= option to 0 and checks the syntax of subsequent steps. Thus, use a STOP statement instead of an ABORT statement to cause SAS to stop the current DATA step and go on to execute the next DATA or PROC step in the job. In display manager and interactive mode, SAS does not set the OBS= option to 0.

The form of the ABORT statement is:

ABORT [**ABEND**][**RETURN**] [n]**;**

You can specify these options in the ABORT statement:

ABEND abnormally terminates the job or session.[1] A SAS data set is not created. For example,

```
ABORT ABEND;
```

Operating systems: All

RETURN causes an immediate normal termination of the SAS job or session with the step return code (condition code) indicating an error. SAS does not enter syntax-check mode or continue to process statements.

Operating systems: All

n sets the step return or condition code to *n*, an integer.[2] For example, if *n* is 255,

```
ABORT 255;
```

sets the step condition code to 255.

When both ABEND and *n* are specified, SAS abnormally terminates the job with the abend code that you specify. For example,

```
ABORT ABEND 255;
```

When both RETURN and *n* are specified, SAS normally terminates the job with the step return or condition code that you specify. For example,

```
ABORT RETURN 16;
```

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, VMS[2]

Before you perform extensive analysis on your data, you can use the ABORT statement to halt execution if the data are not error free. For example,

```
DATA CHECK;
   INPUT SSN 1-9 PAYCODE 11-13;
   IF _ERROR_ THEN ABORT;
   CARDS;
111222333 100
AAABBBCCC 200
444555666 300
;
```

The automatic variable _ERROR_ is set to 1 if errors occur in the data lines. If any errors are found, SAS stops processing observations. In this example the second data line contains invalid data for the variable SSN. When SAS reads this data line, it sets _ERROR_ to 1 and then executes the ABORT statement. Data set CHECK is created with one observation (however, if CHECK is to replace an existing SAS data set named CHECK, the existing data set is not replaced).

Notes

1. **CMS, OS, and VM/PC**: The user completion code is 0999.

   VMS: The completion code is 999. Abnormal termination of a batch job always produces a dump in the VMS batch log, not the SAS log.

   VSE: Abnormal termination always produces a dump.

2. **VMS**: Specifying *n* sets a VMS DCL character symbol called SAS$STATUS to *n*. You can check the return code of any VMS SAS job by using the VMS command

```
SHOW SYMBOL SAS$STATUS
```

ARRAY Statement

Operating systems: see individual statements

Introduction

If you need to process many variables the same way, you can use the ARRAY statement to define the set of variables (either all numeric or all character) as elements of an array. When the array is referenced in SAS statements later in the DATA step, the SAS System substitutes one of the elements of the array for the array reference.

Important note: Version 5 of the SAS System contains a new type of array, the *explicitly subscripted array*. Explicitly subscripted arrays are available under all operating systems and are the preferred method of constructing arrays. The 82 release of the SAS System contained a different type of array, the *implicitly subscripted array*. For compatibility, it is maintained under the operating systems that previously supported it. We recommend using explicitly subscripted arrays wherever possible.

Explicitly Subscripted ARRAY Statement

Operating systems: All

Explicitly subscripted arrays consist of an array name, a reference to the number of elements in the array, and a list of variables. You refer to elements of the array by the array name and the element number (also called the *subscript*). Since you usually want to process more than one element in an array, explicitly subscripted arrays are often placed within iterative DO, DO WHILE, or DO UNTIL groups.

The form of the explicitly subscripted ARRAY statement is

ARRAY *arrayname* {*n*} [**$**][*length*] [*arrayelements*]**;**

where

arrayname names the array. *Arrayname* must be a valid SAS name that is not the name of a SAS variable in the same DATA step.[1]

{*n*} represents the number of elements in the array. *N* can be either a positive integer or an asterisk (*). You can use the asterisk to eliminate counting the number of elements in the array. Here are examples of ARRAY statements with different numbers of elements:[2]

```
ARRAY RAIN{5} JANR FEBR MARR APRR MAYR;
ARRAY MONTH{*} JAN FEB JUL OCT NOV;
ARRAY X{*} _NUMERIC_;
```

All arrays shown in this description are one-dimensional arrays; that is, a single number is used as the subscript.[3]

$ indicates that the elements in the array are character. The dollar sign is not necessary if the elements have been previously defined as character.

length specifies the length of elements in the array that have not previously been assigned a length. See the **LENGTH Statement** later in this chapter for information on the length of SAS variables.

arrayelements names the variables that make up the array. A variable can be an element in more than one arrray. However, you cannot use the name of an explicitly subscripted array or an explicitly subscripted array reference as an element of another array.[4]

Referring to explicitly subscripted arrays You can use an explicitly subscripted array reference anywhere that you can write a SAS expression, including the following SAS statements:[5]

assignment
DO WHILE(*condition*)
DO UNTIL(*condition*)
IF
INPUT
PUT
SELECT

You can also use array references in the arguments of SAS functions. However, an array reference in a function is not a substitute for a variable list since only one element of the array is processed in any given execution of the function.

The ARRAY statement defining the array must appear in a DATA step before any references to that array. An array definition is only in effect for the duration of the DATA step. If you want to use the same array in several DATA steps, you must redefine the elements of the array in each step. Arrays are used in DATA steps and in several SAS procedures that allow programming (such as PROC NLIN and, in the SAS/ETS product, the COMPUTAB procedure).

A reference to an element of an explicitly subscripted array must contain the subscript. The subscript can be any valid SAS expression. Enclose the subscript in braces.[6] For example, you can write the fourth and sixth elements of array TEST as follows:

```
DATA NEW;
   INPUT QA1-QA10 QB1-QB10;
   ARRAY TEST{10} QA1-QA5 QB1-QB5;
   PUT TEST{4}= TEST{6}=;
   CARDS;
data lines
;
```

The PUT statement writes the values of QA4 and QB1, the fourth and sixth elements of array TEST.

Using explicitly subscripted arrays with DO groups You can process explicitly subscripted arrays in an interative DO, DO WHILE, or DO UNTIL group.

To process an array in an iterative DO group:

- create an iterative DO statement in which the starting and stopping values are the beginning and ending elements of the array you want to process.
- use the index variable of the DO statement as the subscript of array references within the group.

In each execution of the DO group, the current value of the index variable is the subscript of the array element being processed. For example, the statements

```
ARRAY DAYS{7} D1-D7;
DO I=1 TO 7;
   IF DAYS{I}=99 THEN DAYS{I}=100;
   END;
```

test the value of variables D1 through D7 in order and, if any of them have a value of 99, change that value to 100.

To process particular elements of an array, specify those elements as the range of the iterative DO statement. For example, you can process selected elements of array DAYS above as follows:

```
DO I=2 TO 4;
DO I=1 TO 7 BY 2;
DO I=3,5;
```

You can also process two or more arrays in an iterative DO group. For example,

```
ARRAY DAYS{7} D1-D7;
ARRAY HOURS{7} H1-H7;
DO I=1 TO 7;
   IF DAYS{I}=99 THEN DAYS{I}=100;
   HOURS[I]=DAYS{I}*12;
   END;
```

You can use the DIM function in the iterative DO statement to return the number of elements in an explicitly subscripted array. The DIM function is useful because it allows you to change the number of elements in an array without respecifying the upper bound of all iterative DO groups that refer to that array. You can also use it when you have specified the number of elements in the array with an asterisk. For example, you can write:

```
DO I=1 TO DIM(DAYS);
DO I=1 TO DIM(DAYS) BY 2;
```

To process explicitly subscripted arrays with a DO WHILE or DO UNTIL statement

- create an index variable for the array before the DO WHILE or DO UNTIL group
- use an array reference in the condition of the DO WHILE or DO UNTIL statement
- use program statements within the group to change the value of the index variable.

This example illustrates a DO WHILE statement:

```
DATA TEST;
   INPUT X1-X5 Y;
   ARRAY T{5} X1-X5;
   I=1;
   DO WHILE(T{I}<Y);
      PUT T{I}= Y=;
      I=I+1;
      END;
   CARDS;
1 2 3 4 5 3
0 2 4 6 8 6
;
```

More examples of explicitly subscripted arrays are given in "Data Step Applications."

Implicitly Subscripted ARRAY Statement

Operating systems: CMS, OS, VM/PC, and VSE

Note: implicit subscripting is an older form of array use that is supported for the operating systems listed for compatibility. We recommend that you use explicitly subscripted arrays instead of implicitly subscripted arrays wherever possible.

Implicitly subscripted arrays consist of an array name, an index variable (either one you supply or a default one), and a list of names (either variable names or names of other implicitly subscripted arrays). They do not contain an explicit reference to the number of elements in the array. You refer to an element of the array by setting the index variable to the number of the element and using the array name in a subsequent statement. Since you usually want to process more than one element of an array, implicitly subscripted arrays are often placed within iterative DO, DO OVER, DO WHILE, or DO UNTIL groups.

The form of the implicitly subscripted ARRAY statement is

ARRAY *arrayname* [(*indexvariable*)][**$**] [*length*] *arrayelements*;

You can specify the items below in the implicitly subscripted ARRAY statement:

arrayname
: names the array. The array name must be a valid SAS name that is not the name of a SAS variable in the same DATA step. When the array name appears in a SAS statement, one of the array elements is substituted for the array name based on the value of the index variable, below.

[(*indexvariable*)]
: gives the name of a variable whose value defines the current element of the array. All implicitly subscripted arrays use an index variable even if you do not specify one in the ARRAY statement. If you do not include an index variable, SAS uses the automatic variable _I_ as the index variable. The value of the index variable can range from 1 to the number of elements in the array.

  The index variable you specify is included in the SAS data set unless it is excluded by a DROP or KEEP statement. The automatic index variable _I_ is not included in the data set.

  If the index variable's value is not an integer, it is truncated to an integer using the rules of the INT function, described in "SAS Functions."

[**$**]
tells SAS that the array you are defining is composed of character elements. The dollar sign ($) is not necessary if the character variables have been previously defined as character in the step.

[*length*]
gives a length value to array elements not previously assigned a length. For example, the statements

```
DATA A;
   INPUT X1 $3. X2 $3.;
   ARRAY ITEM(I) $ 12 X1-X10;
   other SAS statements
```

define a character array ITEM. The first two elements X1 and X2 have lengths of 3, defined in the INPUT statement. The other eight elements of ITEM are given a length of 12 by the length specification in the ARRAY statement.

arrayelements
names the variables that make up the array. Array elements can be the names of variables or other implicitly subscripted arrays and must be either all numeric or all character. You can also use numbered lists such as X1-X10 and the special variable lists _NUMERIC_, _CHARACTER_, and _ALL_ (if all the variables in the data set are the same type). You cannot use lists of the form X--Y, names of explicitly subscripted arrays, or explicitly subscripted array element references. A variable or an implicitly subscripted array can be an element in more than one implicitly subscripted array.

For example, the statement

```
ARRAY QUESTION (I) Q1-Q20;
```

defines an array QUESTION with 20 elements Q1-Q20. Using the name QUESTION in a subsequent SAS statement refers to the *i*th element of QUESTION, where *i* is the current value of the index variable I.

The array elements can be listed in any order, and the names of array elements do not have to be numbered names as in the example above.

The ARRAY statement

```
ARRAY QUIZ (NUM) Q10 Q3-Q5 TEST4;
```

is a valid array definition. Q10 is the first element of the array QUIZ, Q3 is the second, Q4 the third, Q5 the fourth, and TEST4 the fifth. NUM is the index variable. When NUM's value is 1, Q10 is the variable SAS processes when the array name QUIZ appears in a SAS statement.

Referencing arrays You can use an array name in the following SAS statements:

assignment
DO WHILE(*condition*)
DO UNTIL(*condition*)
IF
INPUT
PUT
SELECT

You can also use array names in the arguments of SAS functions. However, an array name in a function is not a substitute for a variable list, since only one element of the array is processed in any given execution of the function.

The ARRAY statement defining the array name must appear in a DATA step before any references to that array. An array definition is only in effect for the duration of the DATA step. If you want to use the same array in several DATA steps, you must redefine the elements of the array in each step. Arrays are used in DATA steps and in several SAS procedures that allow programming (such as PROC NLIN and, in SAS/ETS software, the COMPUTAB procedure).

To refer to an element of an implicitly subscripted array, set the index variable to the index of the element you want and then use the array name in a SAS statement. For example, say that you want to print the value of the eleventh element of the array BIG:

```
DATA ONE;
   INPUT ID X1-X10 Y1-Y10;
   ARRAY BIG (I) X1-X10 Y1-Y10;
   I=11;
   PUT BIG;
   CARDS;
data lines
;
```

The PUT statement writes the value of Y1, the eleventh element of BIG.

Using implicitly subscripted arrays with DO groups You can process an implicitly subscripted array in an iterative DO group, a DO OVER group, a DO WHILE group, or a DO UNTIL group.

The following example processes an implicitly subscripted array with an iterative DO group:

```
DATA TEST;
   INPUT SCORE1-SCORE5;
   ARRAY S SCORE1-SCORE5;
   DO _I_=1 TO 5;
      S=S*100;
      END;
   CARDS;
.95 .88 .57 .90 .65
;
```

In this example the value of each variable SCORE1-SCORE5 is multiplied by 100.

The index variable of the array must also be the index variable of the iterative DO statement. You can process more than one array in an iterative DO group.

The DO OVER statement automatically executes the statements in the DO group for each element of the array unless you specify otherwise. It is equivalent to the statement

```
DO I=1 TO K;
```

where *I* is the index variable of the array, and *K* is the number of elements in the array. The statements in the DO OVER group below change all missing values in the array BIG to the value zero:

```
DATA TWO;
   INPUT ID X1-X10 Y1-Y10;
   ARRAY BIG (I) X1-X10 Y1-Y10;
```

```
DO OVER BIG;
   IF BIG=. THEN BIG=0;
   END;
CARDS;
```

If the implicit array subscript is out of range (less than 1 or greater than the number of elements in the array) when the array name appears, the DATA step stops and SAS prints an error message on the log.

You can process more than one array in a DO OVER group if the arrays have the same number of elements and the same index variable. For example, consider this DATA step:

```
DATA FTOC;
   INPUT F1-F100;
   ARRAY F F1-F100;
   ARRAY C C1-C100;
   DO OVER F;
      C=(F-32)*5/9;
      END;
   CARDS;
```

Each ARRAY statement defines an array with 100 elements. Both arrays use the automatic index variable _I_. The variables in the INPUT statement make up the F array, and the elements in the C array receive values in the DO OVER group. The DO OVER statement here is equivalent to the statement

```
DO _I_=1 TO 100;
```

Either array name F or C could appear in the DO OVER statement since both arrays have the same number of elements and the same index variable.

The following example processes an implicitly subscripted array with a DO WHILE group:

```
DATA TEST;
   INPUT X1-X5 Y;
   ARRAY T X1-X5;
   _I_=1;
   DO WHILE(T<Y);
      PUT T= Y=;
      _I_=_I_+1;
      END;
   CARDS;
1 2 3 4 5 3
0 2 4 6 8 6
;
```

Using arrays as elements of other arrays: multidimensional implicit subscripting

Since an implicitly subscripted array can be an element of another implicitly subscripted array, double and higher-level implicit subscripting is possible. For example, assume your data values consist of answers from three tests, ten questions per test. For each student, there are thirty answers. You can use the ARRAY statements in the DATA step below to change missing values to zeros in these thirty answers:

```
DATA THREE;
   ARRAY TEST1 (J) T1Q1-T1Q10;
   ARRAY TEST2 (J) T2Q1-T2Q10;
   ARRAY TEST3 (J) T3Q1-T3Q10;
```

```
ARRAY ANSWER (K) TEST1-TEST3;
INPUT T1Q1-T1Q10 T2Q1-T2Q10 T3Q1-T3Q10;
DO K=1 TO 3;
   DO J=1 TO 10;
      IF ANSWER=. THEN ANSWER=0;
      END;
   END;
CARDS;
```

The outer DO loop determines which element of ANSWER (array TEST1, TEST2, or TEST3) is being processed. The inner DO loop determines which element in the current array (that is, questions 1 through 10) is being processed.

Notes

1. **CMS, OS, VM/PC, and VSE:** If *arrayname* duplicates a function name, the array is constructed and the function is not available in that DATA step.

2. **AOS/VS, PRIMOS, and VMS:** You can omit {*n*}, and you can also enclose *n* in brackets or parentheses as follows:

   [*n*] | (*n*)

(You should use braces or brackets if they are available on your terminal to avoid possible confusion of array definitions and function calls.)

CMS, OS, VM/PC, and VSE: You must include either *n* or an asterisk in an explicitly subscripted ARRAY statement; omitting both of them creates an implicitly subscripted array. *N* can range from 1 to 65535. (You can also enclose *n* in parentheses; however, using braces avoids possible confusion between array definitions and function calls.)

3. **AOS/VS, PRIMOS, and VMS:** To create a multidimensional array, place the number of elements in each dimension after the array name in the form

   {*n*,...}

(You can also use brackets or parentheses.) *N* is required for each dimension of a multidimensional array.

Reading from right to left, the rightmost dimension represents columns; the next dimension represents rows; and each position farther left represents a higher dimension. Thus, the statement

```
ARRAY X{5,3} SCORE1-SCORE15;
```

defines a two-dimensional array with five rows and three columns.

SAS places variables into a multidimensional array by filling all rows in order, beginning at the upper left corner of the array. In this example variable SCORE1 is element X{1,1}; SCORE2 is X{1,2}; SCORE3 is X{1,3}; SCORE4 is X{2,1}; and SCORE15 is X{5,3}. This example places variables into a two-dimensional array:

```
DATA OVERTIME;
   INFORMAT TIME1-TIME20 TIME5.;
   INPUT ID TIME1-TIME20;
   ARRAY TCHEK{5,4} TIME1-TIME20;
   DO I = 1 TO 5;
      IF INTCK('HOUR',TCHEK{I,1},TCHEK{I,4})>9 THEN
         IF INTCK('HOUR',TCHEK{I,2},TCHEK{I,3})<=1 THEN
            DO;
            PUT ID @10 ' IN ' TCHEK{I,1} TIME5. /
                   @10 'OUT ' TCHEK{I,2} TIME5. /
                   @10 ' IN ' TCHEK{I,3} TIME5. /
```

```
                    @10 'OUT ' TCHEK(I,4) TIME5. ;
                OUTPUT;
                END;
            END;
        CARDS;
129  7:49 11:53  12:01 17:30
     8:01 12:00  12:59 17:05
     8:00 11:59  13:01 17:05
     7:33 10:49  11:50 17:56
     8:00 12:00  13:00 17:00
;
```

CMS, OS, VM/PC, and VSE: You can nest implicitly subscripted arrays to achieve the effect of multidimensional implicitly subscripted arrays. See **Implicitly Subscripted ARRAY Statement** for more information. Multidimensional explicitly subscripted arrays will be available in a future release.

4. **CMS, OS, VM/PC, and VSE:** When *n* is a number, you can omit *arrayelements*. In that case, SAS creates variable names by concatenating the array name and the numbers 1, 2, 3, ... *n*. If a variable name in the series already exists, SAS uses that variable instead of creating a new one. When *n* is an asterisk, you must include *arrayelements*. You can also use the names of implicitly subscripted arrays in the list of array elements; the implicit subscript must contain a valid value when you use that member of the explicitly subscripted array. The array is still one-dimensional.

5. **CMS, OS, VM/PC, and VSE:** You cannot use an explicitly subscripted array reference as an argument of the LABEL or VNAME routine in the CALL statement.

6. **AOS/VS, PRIMOS, and VMS:** You can also enclose the subscript in brackets ([ ]), as in

```
PUT TEST[4]= TEST[6]=;
```

CMS, OS, VM/PC, and VSE: You can also enclose the subscript in parentheses () as in

```
PUT TEST(4)= TEST(6)=;
```

You can use the special array subscript asterisk (*) in an INPUT or PUT statement to refer to all elements of the array. For example, you can write all elements of array TEST as

```
PUT TEST[*]=;
```

You cannot use the asterisk subscript with named input.

Assignment Statement

Operating systems: All

Assignment statements evaluate an expression and store the result in a variable. Assignment statements have the form:

variable=*expression*;

The terms in the assignment have these definitions:

variable names a new variable or an existing variable. *Variable* can be a variable name or an array reference. You can assign values to the special SAS variables FIRST.byvariable, LAST.byvariable, _ERROR_, and _N_; however, the SAS System resets _N_ at the beginning of the DATA step to reflect the number of the current execution of the DATA step, even if you change _N_ during the DATA step. (See "Introduction to the SAS Language" for a discussion of SAS variables, and see the BY statement for a discussion of FIRST. and LAST. byvariables.)

expression is one or more variable names, constants, and function names linked by operators and by parentheses where appropriate. (See "SAS Expressions" and "SAS Functions" for more information.)

For example, the statements

```
DATA ONE;
   INPUT A B;
   X=A+B;
   CARDS;
data lines
;
```

read values of A and B from input lines and create a new variable X that is the sum of A and B values in each observation. If either A or B is missing, X is missing, and the SAS System prints a note on the SAS log to warn you. (See "Missing Values" for details.)

You can modify existing variables by writing assignment statements like this:

```
A=A+B;
```

The sum of A and B replaces the original value of A. Note that the variable A appears on both sides of the statement: the original value of A on the right side is used in evaluating the expression. The result is stored in the variable A.

Variables in assignment statements If a variable appears for the first time in a DATA step as the result variable in an assignment statement, it has the same type (character or numeric) as the result of the expression. If a variable has been defined as numeric and then used as a result variable in an assignment statement with a character expression, SAS performs character-to-numeric conversion on the expression, if possible, and executes the statement. If the conversion is not possible, SAS issues an error message. See "SAS Expressions" for further discussion.

If a variable appears for the first time on the right side of an assignment statement, SAS assumes that it is a numeric variable and that its value is missing. A note is printed on the SAS log that the variable is uninitialized. (A RETAIN statement initializes a variable and can assign it a value, even if the RETAIN statement appears after the assignment statement. See the RETAIN statement for more information.)

The length of a variable appearing for the first time as the result variable in an assignment statement is the length resulting from the first scanning of the statement unless a LENGTH statement specifies a different length. For character variables, the LENGTH statement must come before the assignment statement; for numeric variables, the LENGTH statement can precede or follow the assignment statement. See the LENGTH statement later in this chapter for more information.

Table 4.1 gives the length of a result variable produced by various types of expressions when the length is not explicitly set.

Table 4.1 Length Produced by Various Expression

RESULT VARIABLE TYPE	EXPRESSION	RESULT VARIABLE LENGTH	EXAMPLE
Numeric	Numeric variable	Default numeric length (8 unless other wise specified)	`LENGTH A 4; X=A; *X HAS LENGTH 8;`
Character	Character variable	Length of source variable	`LENGTH A $4; X=A; *X HAS LENGTH 4;`
Character	Character literal	Length of first literal encountered	`X='ABC'; X='ABCDE'; *X HAS LENGTH 3;`
Character	Concatenation of variables	Sum of the lengths of all variables	`LENGTH A $4 B $6 C $2; X=A || B || C; *X HAS LENGTH 12;`
Character	Concatenation of variables and literal	Sum of the lengths of variables and first literal encountered	`LENGTH A $4; X=A||'CAT'; X=A||'CATNIP'; *X HAS LENGTH 7;`

See "SAS Functions" for information on the length of values returned by SAS functions.

More examples that use assignment statements are given in "DATA Step Applications."

ATTRIB Statement

Operating systems: All

The ATTRIB statement in the DATA step allows you to specify the format, informat, label, and length of a variable in a single statement.

The form of the ATTRIB statement is

ATTRIB *variable* [**FORMAT=***format*] [**INFORMAT=***informat*] [**LABEL=***'label'*] [**LENGTH=**[**$**]*length*];

You can specify the following terms in the ATTRIB statement:

variable
names the variable to receive the attributes.[1]

FORMAT=*format*
specifies the format to associate with the preceding variable. The format can be either a SAS format or a format you have created with the FORMAT procedure. See the FORMAT statement later in this chapter for more information on the format attribute, and see "SAS Informats and Formats" for descriptions of SAS formats.

INFORMAT=*informat*
specifies the informat to associate with the preceding variable. See the INFORMAT statement later in this chapter for information on the informat attribute, and see "SAS Informats and Formats" for descriptions of informats.

LABEL=*'label'*
specifies a label to be associated with the preceding variable. See the LABEL statement later in this chapter for information on variable labels.

LENGTH=[$]*length*
specifies the length of the preceding variable. If the variable is a character variable, put a dollar sign ($) in front of the length.

Using the ATTRIB statement in the DATA step permanently associates the attributes with the variables. See the ATTRIB statement in "Statements Used in the PROC Step" for information on assigning attributes for the current PROC step only. You can also assign variable attributes with FORMAT, INFORMAT, LABEL, and LENGTH statements. Any attribute that you have assigned with an ATTRIB statement can be changed in an individual statement and vice versa.

Here are examples of valid ATTRIB statements:

```
ATTRIB X LENGTH=$4 LABEL='TEST VARIABLE';
ATTRIB SALEDAY INFORMAT=MMDDYY. FORMAT=WORDDATE.;
```

Note

1. **AOS/VS, PRIMOS, VMS**: *Variable* can also be a list of names or one of the variable lists described in "Introduction to the SAS Language" except for the special lists _NUMERIC_, _CHARACTER_, and _ALL_. You can also specify several variables with their attributes in succession, as in

```
ATTRIB variable attributes...;
```

BY Statement

Operating systems: All

Introduction

A BY statement is used in a DATA step to control the operation of a SET, MERGE, or UPDATE statement and to set up special grouping variables. See "SAS Statements Used in the PROC Step" for information on the BY statement in the PROC step.

The BY statement has the form:

BY [**DESCENDING**] *variable* ... [**NOTSORTED**] **;**

where

variable
names each variable by which the data set is sorted. The list of variables defines the data set's BY groups.

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 with the same BY values are grouped together, but the BY values are not necessarily sorted in alphabetical or numerical order. The NOTSORTED option can appear anywhere in the BY statement. The option can only be used with a SET statement which reads a single data set.

For example, the statement

```
BY DESCENDING X Y;
```

specifies that the data set is sorted in descending order of the values of X and, within each X value, in ascending order of Y. The statement

```
BY DESCENDING X DESCENDING Y;
```

specifies that the data set is sorted in descending order of the values of both X and Y.

Suppose data set CLASS contains observations with a variable DAY whose values are the three-character abbreviation for the day of the week. All the observations with the same DAY value are grouped together in the data set, but the values are in calendar order rather than alphabetical order. You can use the statement

```
BY DAY NOTSORTED;
```

with a SET statement to read the CLASS data set.

When you use a BY statement without specifying the NOTSORTED option, the data set used as input to the step need not have been previously sorted by the SORT procedure. However, the data set must be in the same order as though the SORT procedure had been used.

If neither DESCENDING or NOTSORTED is specified, the observations in the data set must be arranged in ascending order of the BY variables' values.

BY Groups

These statements:

```
PROC SORT DATA=DEGREES;
   BY STATE CITY MONTH;
```

produced the SAS data set below using the SORT procedure. The data set DEGREES is sorted by STATE, CITY within STATE, and MONTH within CITY. The fourth variable in the data set, DEGDAY, represents winter degree days.

STATE	CITY	MONTH	DEGDAY	
NC	CHARLOTTE	1	716	
NC	CHARLOTTE	2	588	
NC	CHARLOTTE	3	461	
				CITY break
NC	RALEIGH	1	760	
NC	RALEIGH	2	638	
NC	RALEIGH	3	502	
				STATE break-CITY break
VA	NORFOLK	1	760	
VA	NORFOLK	2	661	
VA	NORFOLK	3	532	
				CITY break
VA	RICHMOND	1	853	
VA	RICHMOND	2	717	
VA	RICHMOND	3	569	
				STATE break-CITY break

The BY groups in the data set, from the largest grouping to the smallest, are represented by different values for STATE, CITY within STATE, and MONTH within CITY within STATE. In the data set above, NC and VA observations break the data set into two STATE BY groups; CHARLOTTE and RALEIGH observations divide the NC observations into CITY BY groups; NORFOLK and RICHMOND observations divide VA observations into CITY BY groups; and within CITY each observation, with its different value for MONTH, is itself a MONTH BY group.

FIRST. and LAST.byvariables

You can process the data set in BY groups if you include a BY statement after the SET, MERGE, or UPDATE statement in the DATA step.[1]

When you are processing a data set in BY groups, you may want to identify which observation is the first or the last in a particular BY group. SAS keeps track of the first and last observations in all BY groups by creating two variables, FIRST.*byvariable* and LAST.*byvariable*, for each variable in the BY statement. The FIRST. and LAST.*byvariables* let you know when you are processing the first or last observation in a BY group. For example, if you use a BY statement containing variables STATE, CITY, and MONTH, SAS creates the variables FIRST.STATE, LAST.STATE, FIRST.CITY, LAST.CITY, FIRST.MONTH, and LAST.MONTH. FIRST.*byvariables* and LAST.*byvariables* are available in the DATA step for use in program statements but are not added to the data set being created. To display the FIRST. and LAST.*byvariables*, use a PUT statement.

If an observation is the first in a BY group, then FIRST.*byvariable* has the value 1. For all observations in the BY group except the first, FIRST.*byvariable* is 0. If an observation is the last in a BY group, LAST.*byvariable* is 1. For all observations in the BY group except the last, LAST.*byvariable* is 0.

The data set DEGREES is shown with values of the FIRST. and LAST.*byvariables*:

				FIRST.*byvariables*			LAST.*byvariables*		
STATE	CITY	MONTH	DEGDAY	STATE	CITY	MONTH	STATE	CITY	MONTH
NC	CHARLOTTE	1	716	1	1	1	0	0	1
NC	CHARLOTTE	2	588	0	0	1	0	0	1
NC	CHARLOTTE	3	461	0	0	1	0	1	1
									CITY break
NC	RALEIGH	1	760	0	1	1	0	0	1
NC	RALEIGH	2	638	0	0	1	0	0	1
NC	RALEIGH	3	502	0	0	1	1	1	1
								STATE break	CITY break
VA	NORFOLK	1	760	1	1	1	0	0	1
VA	NORFOLK	2	661	0	0	1	0	0	1
VA	NORFOLK	3	532	0	0	1	0	1	1
									CITY break
VA	RICHMOND	1	853	0	1	1	0	0	1
VA	RICHMOND	2	717	0	0	1	0	0	1
VA	RICHMOND	3	569	0	0	1	1	1	1
								STATE break	CITY break

Notice that each MONTH BY group consists of 1 observation. Thus, FIRST.MONTH and LAST.MONTH are both 1 in each observation. Notice also that when the FIRST.*byvariable* is 1 for a particular variable, the FIRST.*byvariable* is also 1 for all variables following it in the BY statement. Thus, when FIRST.STATE is 1 for an observation, FIRST.CITY and FIRST.MONTH are also 1 for that observation. The same is true for LAST.*byvariables*.

For example, the data set below is sorted by WEEK and by DAY within WEEK. Notice the values of the FIRST. variables:

OBS	WEEK	DAY	FIRST.WEEK	FIRST.DAY
1	1	MONDAY	1	1
2	1	TUESDAY	0	1
3	1	TUESDAY	0	0
4	2	TUESDAY	1	1
5	2	TUESDAY	0	0
6	2	WEDNESDAY	0	1

Although in observation 4, the DAY value is not the first with the value 'TUESDAY', it is the first value of TUESDAY within WEEK=2. That is, since FIRST.WEEK is 1, FIRST.DAY is 1.

BY Statement with MERGE

When a BY statement is used in a DATA step with a MERGE statement, the data sets listed in the MERGE statement are joined by matching values of the variables listed in the BY statement. FIRST. and LAST.*byvariables* are created for each variable listed in the BY statement. See the **MERGE Statement** for more information.

The statements below perform a match-merge, matching on STATE, and CITY within STATE. SAS data sets YR84 and YR85 have been previously sorted by STATE and CITY using the SORT procedure. The LAST.CITY variable is used in the IF statement, and the last observation in each CITY BY group is output to data set SINGLE. All observations are then output to data set NEW:

```
DATA NEW SINGLE;
   MERGE YR84 YR85;
   BY STATE CITY;
   IF LAST.CITY THEN OUTPUT SINGLE;
   OUTPUT NEW;
```

The NOTSORTED option cannot be used in a BY statement in a DATA step with the MERGE statement; the DESCENDING option can be used if the observations being merged are sorted in descending order of the variables mentioned in the BY statement.

See the **MERGE Statement** description for more information.

BY Statement with UPDATE

A BY statement must appear in a DATA step with an UPDATE statement to identify the matching variable or variables to be used in the update. See the **UPDATE Statement** later in this chapter for more information. FIRST. and LAST.*byvariables* are created, and the NOTSORTED option **cannot** be used. "DATA Step Applications" contains more examples using a BY statement in update applications.

See the **UPDATE Statement** description for more information.

BY Statement with SET

When a BY statement is used in a DATA step that includes a SET statement with one data set name, the data set being read is processed just as though a BY statement were not included. The difference is that the BY statement causes FIRST.

and LAST.*byvariables* to be created and produces an error message if the data set is not sorted (and the NOTSORTED option is not present).

For example, the DATA step below creates a SAS data set named SUBSET, which includes only the first observation in each STATE BY group from data set DEGREES:

```
DATA SUBSET;
   SET DEGREES;
   BY STATE;
   IF FIRST.STATE;
```

Only STATE, the primary BY group, is used in the BY statement since only the FIRST.STATE variable is needed.

When a BY statement is used in a DATA step with a SET statement, both the NOTSORTED and DESCENDING options can be used.

See the **SET Statement** description for more information.

Note

1. **AOS/VS, PRIMOS, and VMS**: the BY statement must follow the SET, MERGE, or UPDATE statement.

   CMS, OS, VM/PC, and VSE: the BY statement can follow or precede the SET, MERGE, or UPDATE statement.

CALL Statement

Operating systems: All

The purpose of the CALL statement in the DATA step is to invoke or call a routine. The routine is called each time the CALL statement is executed.

The form of the CALL statement is

CALL *routine*(*parameter*, ...);

where

routine names the routine you want to call. Individual routines are described below.

parameter is a piece of information, usually a SAS variable name, to be passed to the routine.

Routines that can be used with the CALL statement include:

external names a user-written routine. For information on how to write a call routine, see the *SAS Programmer's Guide*.

Operating systems: CMS, OS, VM/PC, and VSE

LABEL returns the label of a variable. The form of the LABEL routine is

CALL LABEL(*variable1*,*variable2*);

The LABEL routine assigns the label of the variable specified as *variable1* to the character variable specified as *variable2*. If *variable1* does not have a label, the variable name is assigned as the value of *variable2*. Since a variable label can be up to forty characters long, the length of *variable2* should be at least forty characters to avoid truncating any variable labels. Do not use explicitly subscripted array references as arguments of the LABEL routine.

For example, the statements

```
DATA ONE;
   LENGTH X $40;
   X=' ';
   SCORE=23;
   LABEL SCORE='PRE-TEST SCORE';
   CALL LABEL(SCORE,X);
RUN;
```

assign X the value PRE-TEST SCORE.

Operating systems: CMS, OS, VM/PC, and VSE

randomnumber-function specifies one of the following SAS random number functions:

RANBIN RANPOI
RANCAU RANTBL
RANEXP RANTRI
RANGAM RANUNI
RANNOR

See "SAS Functions" for a description of these functions.

Operating systems: All

SYMPUT specifies the SYMPUT function, which assigns a value from the DATA step to a macro variable. The SYMPUT function is discussed in "SAS Functions" and "SAS Macro Language."

Operating systems: CMS, OS, VM/PC, and VSE

VNAME assigns the name of a variable as the value of another variable. The form of the VNAME routine is

CALL VNAME(*variable1***,***variable2***);**

where *variable1* and *variable2* are names of SAS variables. The VNAME routine assigns the name of *variable1* as the value of *variable2*. Since a SAS variable name can contain up to eight characters, the length of *variable2* should be at least 8. Do not use explicitly subscripted array references as arguments of the VNAME routine.

The following example illustrates the VNAME routine:

```
DATA B;
   LENGTH NNN $8;
   NNN=' ';
   SALARY=1500;
   CALL VNAME(SALARY,NNN);
```

The value of variable NNN is SALARY.

Operating systems: CMS, OS, VM/PC, and VSE

CARDS and CARDS4 Statements

Operating systems: All

If you are entering your data in the job stream with your SAS program, the CARDS statement comes before the data lines to signal to the SAS System that the data follow.[1] The form of the CARDS statement is

CARDS;
data lines
;

If you use a CARDS statement, it must be the last statement in your DATA step, and it must be followed immediately by data lines. Data lines following a CARDS statement have a maximum length of 80 columns.

SAS recognizes the end of the data lines when it sees a semicolon. The first line after the last data line should be either a null statement (a line containing a single semicolon) or another SAS statement ending with a semicolon on the same line:

```
   CARDS;
data lines
;
```

You must use an INPUT statement to read values from data lines following a CARDS statement. For example,

```
DATA ENTRIES;
   INPUT X Y;
   CARDS;
data lines
;
```

The CARDS statement is for data lines entered following a DATA step (also called *in-stream data*); if your input lines are on disk or tape, use instead the INFILE statement described later in this chapter.

You can use INFILE statement options in reading data following a CARDS statement by using an INFILE statement with a fileref of CARDS together with a CARDS statement. For example,

```
DATA TEMP;
   INFILE CARDS MISSOVER;
   INPUT X Y Z;
   CARDS;
1 10 100
2 20
3 30 300
;
```

Only one CARDS statement can be used in a DATA step. If you want to enter two sets of data, either use two DATA steps or two INFILE statements, one for each set of data. (See the INFILE statement for an example.)

Since a semicolon signals the end of the data lines to SAS, if the data lines following the CARDS statement happen to contain semicolons, you must substitute the CARDS4 statement for the CARDS statement. The form of the CARDS4 statement is

CARDS4;
data lines
;;;;

After the last data line, put a line consisting of four semicolons in columns 1-4 to signal the end of the data. For example,

```
DATA _NULL_;
   INPUT NUMBER CITATION & $50.;
   FILE PRINT;
   PUT NUMBER @3 CITATION;
   CARDS4;
1  SMITH, 1982
2  ALLEN ET AL., 1975; BRADY, 1983
3  BROWN, 1980; LEWIS, 1974; WILLIAMS, 1972
;;;;
another DATA or PROC statement
```

Note

1. **AOS/VS, PRIMOS, and VMS**: You cannot use a statement label on a CARDS or CARDS4 statement.

CMS, OS, VM/PC, and VSE: You can use a statement label on a CARDS or CARDS4 statement.

DATA Statement

Operating systems: All

Introduction

The DATA statement begins a DATA step and provides a name for the SAS data set or data sets being created. SAS data sets are described in detail in "SAS Files."

The form of the DATA statement is

DATA [[*SASdataset*[(*dsoptions*)]]...];

The terms in the DATA statement are described below:

SASdataset

names one or more SAS data sets being created in the DATA step. SAS data set names must conform to the rules for SAS names as well as to the file naming rules for your operating system. See "Introduction to the SAS Language" for a description of SAS names, and see the SAS companion for your operating system for your system's file naming rules. [1]

The data set name given in the DATA statement may be a one-word name (for example, FITNESS), a two-word name (for example, OUT.FITNESS), or one of the special SAS names _NULL_, _DATA_, or _LAST_.

A DATA statement may name one data set:

```
DATA FITNESS;
DATA OUT.FITNESS;
DATA _NULL_;
```

or more than one:

```
DATA YEAR1 YEAR2 YEAR3;
DATA MALES FEMALES;
DATA LIBRARY.TOTAL ERRORS;
```

You may omit the data set name entirely:

```
DATA;
```

Then the SAS System names the data set using the DATA*n* convention (see below). "SAS Files" gives more information on naming SAS data sets.

(*dsoptions*)
gives the SAS System more information about the data set being created. Data set options in parentheses follow the name of the data set in the DATA statement to which they apply.

For example, LABEL= allows you to specify a label of up to forty characters for the data set. To give the FITNESS data set the label HEALTH CLUB DATA, use the statement:

```
DATA FITNESS (LABEL='HEALTH CLUB DATA');
```

The use of these options is not confined to the DATA statement. Data set options can be associated with a data set name where it occurs in other SAS statements (except the OUTPUT statement in the DATA step).

Options most commonly used in the DATA statement are summarized below. Data set options in this list that do not apply under a particular operating system are accepted but ignored under that operating system.

DROP= lists the variables not to be included in the data set.

Operating systems: All

GEN= specifies the number of generations of history information that are stored with the data set.

Operating systems: CMS, OS, VM/PC, VSE

KEEP= lists the variables to be included in the data set.

Operating systems: All

LABEL= provides the data set with a label that is printed along with the name in the output of the CONTENTS procedure.

Operating systems: All

PROTECT= specifies a write-only password for the data set. The password should be a valid SAS name.

Operating systems: CMS, OS, VM/PC, VSE

READ= specifies a read-and-write password for the data set. The password should be a valid SAS name.

Operating systems: CMS, OS, VM/PC, VSE

RENAME= changes variable names.

Operating systems: All

TYPE= specifies the data set's type. Use TYPE= only when the data set contains specially structured data such as a covariance matrix.

Operating systems: All

Refer to "SAS Files" for a complete description of these and other data set options as well as further discussion of the naming conventions mentioned below.

Choosing a Data Set Name in the DATA Statement

You should name each SAS data set in a SAS program that creates several data sets. Although some SAS users omit the data set name for very simple SAS jobs, you can keep track of your data sets more easily if you give each of them a unique name.

You must include a two-word data set name when you are storing your SAS data set on disk or tape for use in a future SAS job. (The only exception is when you specify the USER= option; see "SAS Files" for discussion.)

Default data set name If you omit a data set name from the DATA statement as shown below:

```
DATA;
```

SAS still creates a data set. The data set is named automatically: the first such data set created in a job is called DATA1, the second is called DATA2, and so on through the job. This is called the DATA*n* naming convention and is equivalent to specifying:

```
DATA _DATA_;
```

where _DATA_ is a special SAS variable indicating that SAS is to use DATA1, DATA2, and so on.

One-word data set name All SAS data set names have two parts separated by a period. The first part (called the *first-level name* or *libref*) identifies the location where the data set is stored; the second part (the *second-level name*) identifies the particular data set. Unless you plan to store your SAS data set on disk or tape, you can give your data set a one-word name. For example, to begin creating a SAS data set named FITNESS, you use the statement

```
DATA FITNESS;
```

When you give a one-word name like FITNESS, SAS uses that name as the second part of the data set name and uses WORK (or the libref specified by the USER= option) as the first part. SAS notes and messages refer to the data set as WORK.FITNESS.

When the first-level name is WORK, the data set is temporary; it is only available during the current job or session.

Two-word data set name Use a two-word data set name if you want to store your SAS data set on disk or tape. For example,

```
DATA OUT.FITNESS;
```

creates a SAS data set named FITNESS in the location on disk or tape referenced by the libref OUT. See "SAS Files" for details on librefs.

If you have used a two-word data set name in the DATA step to create a data set, you must continue to use the two-word name if you refer to the data set later in your SAS job.[2] For example, if you use OUT.FITNESS as shown above to create a SAS data set on disk or tape and you want to refer to the data set in a SET statement later in the job, use the statement

```
SET OUT.FITNESS;
```

If you use the statement

```
SET FITNESS;
```

SAS looks for WORK.FITNESS since it always attaches the name WORK to one-word names.

Using Special SAS Data Set Names

Data set name _NULL_ You can give the special SAS data set name _NULL_ in the DATA statement whenever you want to execute the statements in a DATA step but do not want to create a SAS data set. For example, if you are using a DATA step to print a report with PUT statements based on the values contained in a SAS data set, you probably do not want to create another SAS data set containing the same data. Using _NULL_ means that SAS goes through the DATA step just as if it is creating a data set, but it does not write any observations; thus, the DATA step uses fewer computer resources.

Data set name _LAST_ _LAST_ refers to the name of the most recently created SAS data set. If a DATA statement includes the name _LAST_, the data set created has the same name as the most recently created data set and replaces the earlier data set.

Using More Than One Data Set Name

Subsets of observations When you are creating several data sets in one DATA step and want each data set to include a subset of the observations that the DATA step is processing, you can use the OUTPUT statement to add the appropriate observations to each data set. For example, you may use these statements:

```
DATA YEAR82 YEAR83 YEAR84;
   INPUT YEAR X1-X20;
   IF YEAR=1982 THEN OUTPUT YEAR82;
   ELSE IF YEAR=1983 THEN OUTPUT YEAR83;
   ELSE IF YEAR=1984 THEN OUTPUT YEAR84;
   CARDS;
data lines
;
```

The YEAR82 data set contains only those observations where the YEAR value is 1982; the YEAR83 data set contains only those observations where the YEAR value is 1983; the YEAR84 data set contains only those observations with YEAR= 1984. Observations with any other year values are not written to any data set.

Subsets of variables When you want to create several data sets containing different groups of variables, you can use the KEEP= or DROP= data set options after each data set name in the DATA statement.

For example, say that you want the data set YEAR82 to include, in addition to the variable YEAR, variables X1-X5; data set YEAR83 to include YEAR and X6-X20; and data set YEAR84 to include YEAR and all of the variables X1-X20 except X13:

```
DATA YEAR82(KEEP=YEAR X1-X5)
     YEAR83(KEEP=YEAR X6-X20)
     YEAR84(DROP=X13);
```

You can use either the KEEP= or DROP= option. The advantage of DROP= is that you can write a shorter list when you are dropping fewer variables than you are keeping.

Notes

1. **CMS, VM/PC, VMS**: A SAS data set name cannot contain an underscore; however, the special SAS names _DATA_, _NULL_, and _LAST_ are available.

OS: A SAS data set name can contain an underscore, although the OS rules for OS data set names do not allow an underscore.

2. **AOS/VS**, **PRIMOS**, **VMS**: You can continue to refer to the data set with a one-word name if you have included the libref for the data set in a LIBSEARCH statement or if you have specified its libref as the value of the USER= option.

CMS, **OS**, **VM/PC**, **VSE**: You can continue to refer to the data set with a one-word name if you have specified its libref as the value of the USER= option.

DELETE Statement

Operating systems: All

A DELETE statement tells the SAS System to stop processing the current observation. The observation is not automatically written to any data set being created, and the SAS System returns to the beginning of the DATA step for another execution.

The DELETE statement has the form:

DELETE;

Here is an example:

```
DATA JRHIGH;
   INPUT SSN 1-9 GRADE 10 PRETEST1 14-16 PRETEST2 18-20;
   IF GRADE<7 THEN DELETE;
   TOTAL=PRETEST1+PRETEST2;
   CARDS;
data lines
;
```

In this example, observations with a GRADE value less than 7 are not added to the SAS data set JRHIGH. The assignment statement is not executed for those observations.

When a DELETE statement is executed, SAS immediately stops executing program statements for the current observation. SAS returns to the statement following the DATA statement and begins executing the statements that follow. The DELETE statement is usually used as the THEN clause in an IF statement or as part of a conditionally executed DO group. If DELETE is executed for every observation, the new data set will have no observations.

In general, use the DELETE statement when it is easier to specify a condition for excluding observations from the data set. Use the subsetting IF statement (discussed later in this chapter) when it is easier to specify a condition for including observations.

DO Statement

Operating systems: See individual statements

Introduction

The DO statement specifies that the statements following the DO are to be executed as a unit until a matching END statement appears. The statements between the DO and the END statements are called a DO group. Any number of DO groups can be nested. The SAS System has several forms of the DO statement:

```
DO;
iterative DO;
DO OVER;
DO WHILE;
DO UNTIL;
```

The number of times the DO group is executed depends on the form used.

Simple DO Statement

Operating systems: All

The form of a simple DO statement is

DO;
  more SAS statements
  END;

A simple DO statement is often used within IF-THEN/ELSE statements to designate a group of statements to be executed depending on whether the IF condition is true or false.

For example, the statements

```
IF X>5 THEN DO;
   Y=X*10;
   PUT X= Y=;
   END;
Z=X+3;
```

specify that the statements between DO and the END (the DO group) are to be performed only if X is greater than 5. If X is less than or equal to 5, statements in the DO group are skipped and the next statement executed is the assignment statement.

The statements

```
IF X>5 THEN Y=X*10;
IF X>5 THEN PUT X= Y=;
Z=X+3;
```

produce the same result, but the program is less efficient because the IF expression is evaluated twice.

Iterative DO Statement

Operating systems: All

Iterative execution of a DO group can be specified by using an index variable in the DO statement. The iterative DO statement causes the statements between the DO and the END to be executed repetitively based on the value of the index variable. The form of the iterative DO statement is

DO *indexvariable*=*start* [**TO** *stop* [**BY** *increment*][**WHILE** | **UNTIL**(*expression*)]]...;
SAS statements
END;

You can use these terms in the iterative DO statement:

indexvariable
names a variable whose value governs execution of the DO group. After the final execution of the DO group, the value of the index variable is

start + *increment**(INT((*stop*-*start*)/*increment*)+1) .

Unless dropped, the index variable is included in the data set being created.

start
stop
specify numbers or expressions that yield numbers to control the number of times the statements between DO and END are executed. The DO group is executed first with *indexvariable* equal to *start*.

When both *start* and *stop* are present, execution continues (based on *increment*'s value) until *indexvariable* is greater than *stop*. When only *start* is present, execution continues (based on *increment*'s value) until a statement directs execution out of the loop. If neither *stop* nor *increment* is specified, the group executes once. Values of *start* and *stop* are evaluated prior to the first execution of the loop.

For example, each of these DO groups is executed ten times:

```
DO I=1 TO 10;
   SAS statements
   END;

DO I=1 BY 1;
   SAS statements
   IF I=10 THEN GO TO F;
   END;
F:PUT 'FINISHED';
```

You can also replace *start* and *stop* with a series of values separated by commas. The values can be numeric or character. For example:

```
DO COUNT=2,3,5,7,11,13,17;
```

executes the DO group for COUNT=2,3,5,7,11,13,17. The statement

```
DO MONTH='JAN','FEB','MAR';
```

executes the DO group three times, once for each value JAN, FEB, and MAR of the index variable MONTH. See "DATA Step Applications" for an example of a DO group with a character index variable.

increment

specifies a number or an expression that yields a number to control incrementing of the value of *indexvariable*. The value of *increment* is evaluated prior to the execution of the loop. For example,

```
DO COUNT=2 TO 8 BY 2;
```

causes the DO group to be executed for COUNT=2,4,6,8. The value of COUNT after the final execution of the DO group is 10.

If no increment is specified, the index variable is incremented by 1. If *increment* is positive, then *start* must be the lower bound and *stop*, if present, must be the upper bound for the loop. If *increment* is negative, then *start* must be the upper bound and *stop*, if present, must be the lower bound for the loop.

expression

is any expression. The rules for using a WHILE or UNTIL clause are the same as for the DO WHILE and DO UNTIL statements described later in this section. For example,

```
DO I=1 TO 10 WHILE(X<Y);
DO I=2 TO 20 BY 2 UNTIL((X/3)>Y);
DO I=10 TO 0 BY -1 WHILE(MONTH='JAN');
```

A number of clauses separated by commas can be included in the statement.[1] For example, the statement

```
DO COUNT=2 TO 8 BY 2,11,13 TO 16;
```

causes the DO group to be executed for the following values of COUNT: 2,4,6,8, 11,13,14,15, and 16.

Here are some other examples of valid iterative DO statements:

```
DO I=5;
DO I=1 TO N;
DO I=N TO 1 BY -1;
DO I=K+1 TO N-1;
DO I=1 TO K-1, K+1 TO N;
DO I=2,3,5,7,11,13,17;
DO I=.1 TO .9 BY .1, 1 TO 10 BY 1, 20 TO 100 BY 10;
DO I='SATURDAY','SUNDAY';
DO I='01JAN85'D,'25FEB85'D,'18APR85'D;
```

The values of *start*, *stop*, and *increment* are evaluated before the statements in the DO group are executed. Any changes to the upper bound or increment made within the DO group do not affect the number of iterations. For example, consider this DATA step:

```
DATA ITERATE1;
   INPUT X;
```

```
STOP=10;
DO I=1 TO STOP;
   *ALWAYS 10 ITERATIONS;
   Y=X*NORMAL(0);
   OUTPUT;
   *CHANGING VALUE OF STOP DOES NOT STOP EXECUTION OF GROUP;
   IF Y>25 THEN STOP=I;
   END;
CARDS;
```

Setting STOP to the current value of I does not stop execution of the DO loop. You can, however, change the value of the index variable. Thus, you can stop execution of the loop either by setting I to STOP's value or by using a GO TO statement to jump to a statement outside the loop. For example,

```
DATA ITERATE2;
   INPUT X;
   STOP=10;
   DO I=1 TO STOP;
      *NUMBER OF ITERATIONS RANGES FROM 1 TO 10;
      Y=X*NORMAL(0);
      OUTPUT;
      *EITHER STATEMENT STOPS EXECUTION OF GROUP WHEN Y>25;
      [IF Y>25 THEN I=STOP;] or[IF Y>25 THEN GO TO OUT;]
      END;
   [OUT:;]
   CARDS;
```

Consider these statements:

```
DATA CREATEX;
   INPUT X Y Z;
   IF X=. THEN DO K=1 TO 25;
      X=UNIFORM(0)*100;
      OUTPUT;
      END;
   CARDS;
```

Each time X is missing, 25 observations are output, each with the values of Y and Z from the current data line and with randomly generated values of X. Each time the value of X is not missing, no observation is output.

Array processing with the iterative DO statement You can use an iterative DO statement to execute the statements in a DO group for some or all of the variables in an array. The array can have an explicit or an implicit subscript.

To process an array in an iterative DO group, create an iterative DO statement in which *start* and *stop* are the beginning and ending element numbers of the array you want to process. Within the DO group, refer to elements of the array by their index value. Each of the following examples recodes the value 99 to 100 for any of the variables D1 through D7 that have a value of 99:

```
*USING EXPLICITLY SUBSCRIPTED ARRAY STATEMENT;
*OPERATING SYSTEMS: ALL;
ARRAY DAYS{7} D1-D7;
DO I=1 TO 7;
   IF DAYS{I}=99 THEN DAYS{I}=100;
   END;
```

```
*USING IMPLICITLY SUBSCRIPTED ARRAY STATEMENT;
*OPERATING SYSTEMS: CMS, OS, VM/PC, VSE;
ARRAY DAYS(INDX) D1-D7;
DO INDX=1 TO 7;
   IF DAYS=99 THEN DAYS=100;
   END;
```

See the **ARRAY Statement** earlier in this chapter for more information.

DO OVER Statement

Operating systems: CMS, OS, VM/PC, and VSE

You can execute the statements in a DO group for the elements in an implicitly subscripted array with the DO OVER statement.

The form of the DO OVER statement is

DO OVER *arrayname*;
SAS statements;
END;

where

arrayname
specifies an array that has been previously defined in an implicitly subscripted ARRAY statement (see the description of the **ARRAY Statement** earlier in this chapter for details).

The DO OVER statement automatically executes the statements in the DO group for each element of the array unless you specify otherwise. It is equivalent to the statement

```
DO I=1 TO K;
```

where I is the index variable of the array, and K is the number of elements in the array.

For example, in this DATA step:

```
DATA TEST;
   INPUT SCORE1-SCORE5;
   ARRAY S SCORE1-SCORE5;
   DO OVER S;
     S=S*100;
     END;
   CARDS;
.95 .88 .57 .90 .65
;
```

the DO OVER statement causes the values of SCORE1 through SCORE5 to be multiplied by 100.

See the **ARRAY Statement** earlier in this chapter for more information.

DO WHILE Statement

Operating systems: All

You can execute the statements in a DO group repetitively while a condition holds using the DO WHILE statement. The form of the DO WHILE statement is

DO WHILE(*expression*);

where

expression
is any expression (see "SAS Expressions" for details). The expression is evaluated at the top of the loop before the statements in the DO group are executed. If the expression is true, the DO group is executed.

These statements repeat the loop as long as N is less than 5. There are 5 iterations in all (0, 1, 2, 3, 4):

```
N=0;
DO WHILE(N LT 5);
   PUT N=;
   N+1;
   END;
```

See "Data Step Applications" and the **ARRAY Statement** for more examples.

DO UNTIL Statement

Operating systems: All

The DO UNTIL statement, like the DO WHILE statement, executes the statements in a DO loop conditionally. The DO UNTIL evaluates the condition at the bottom of the loop rather than at the top (as does DO WHILE). Thus, the statements between DO and END are always executed at least one time. The form of the DO UNTIL statement is

DO UNTIL(*expression*)**;**

where

expression
is any expression (see "SAS Expressions" for details). The expression is evaluated at the bottom of the loop after the statements in the DO group have been executed. If the expression is true, the DO group is not executed again. The DO group is always executed at least once.

These statements repeat the loop until N is greater than or equal to 5. There are 5 iterations in all (0, 1, 2, 3, 4):

```
N=0;
DO UNTIL(N>=5);
   PUT N=;
   N+1;
   END;
```

See "Data Step Applications" for additional DO UNTIL examples.

Note

1. **AOS/VS, PRIMOS, and VMS**: In an iterative DO statement with more than one clause, all clauses are evaluated before the execution of the DO statement. A WHILE or UNTIL specification affects only the clause in which it is located and can be in any clause except the first.

CMS, OS, VM/PC, and VSE: In an iterative DO statement with more than one clause, each clause is evaluated prior to the execution of that clause. A

WHILE or UNTIL specification affects all clauses and must be the last clause in the statement.

DROP Statement

Operating systems: All

You can use the DROP statement in a DATA step to specify variables that are not to be included in the SAS data set or sets being created. A DROP statement in a DATA step applies to all the SAS data sets being created in the step. To drop variables selectively when multiple data sets are being created, use the DROP= data set option with each data set name. (See "SAS Files" for more information about data set options.)

Although variables appearing in a DROP statement are not included in any SAS data set being created, these variables can be used in program statements. The DROP statement can appear anywhere in the DATA step with the same effect since it is not an executable statement.

The form of the DROP statement is

DROP *variables;*

where

variables specifies the variables you want omitted from the data set(s) being created. Any form of variable list may be used (see "Introduction to the SAS Language" for details). Do not abbreviate the variable names.

Here is an example:

```
DATA PARTS;
   INPUT NAME $ PARTA PARTB;
   TEST=PARTA+PARTB;
   DROP PARTA PARTB;
   CARDS;
data lines
;
```

The variable TEST is computed for each observation by adding the values of PARTA and PARTB. The DROP statement tells the SAS System not to include the variables PARTA and PARTB in the new data set.

The effect of the DROP statement is the reverse of the KEEP statement's effect. To save writing, the DROP statement is preferred if fewer variables are being dropped than kept. Do not use both DROP and KEEP statements in the same DATA step.

When both RENAME and DROP statements are used in a DATA step, the DROP statement is applied first. This means that the old name should be used in the DROP statement.

Note: in Version 5, the DROP statement is available only in the DATA step. To exclude variables from a PROC step, use the DROP= data set option.

END Statement

Operating systems: All

The END statement is the last of the SAS statements that make up a DO group or a SELECT group. The form of the END statement is

END;

An END statement must end every DO group and SELECT group in your SAS job. For example, a simple DO group and a simple SELECT group are shown below:

```
DO;
  more SAS statements
  END;


SELECT(expression);
   WHEN(expression) SAS statement;
   OTHERWISE SAS statement;
   END;
```

See the DO statement and SELECT statement descriptions elsewhere in this chapter for examples using END.

ERROR Statement

Operating systems: CMS, OS, VM/PC, and VSE

You can use the ERROR statement to identify data lines whose values meet the condition you have specified as an error. When an ERROR statement is executed, the SAS System sets the automatic variable _ERROR_ to 1 and, optionally, prints on the SAS log a message you specify. The action of the ERROR statement is equivalent to an assignment statement setting _ERROR_ to 1, a FILE LOG statement, and (if you specify a message) a PUT statement including a quoted string.

The form of the ERROR statement is

ERROR [*message*];

where

message can include character literals, variable names, formats, and pointer controls just like those described in the PUT statement description later in this chapter.

When the value of _ERROR_ is 1, SAS writes the data lines corresponding to the current observation on the SAS log.

For example, these statements:

```
DATA CHECK;
   INPUT TYPE $ AGE;
   FILE OUT;
   PUT TYPE 1-5 AGE 6-7;
   IF TYPE='TEEN' & AGE>19 THEN
      ERROR 'TYPE AND AGE DON''T MATCH ' AGE=;
   CARDS;
ADULT 30
TEEN  25
TEEN  14
;
```

are equivalent to the statements:

```
DATA CHECK;
   INPUT TYPE $ AGE;
   FILE OUT;
   PUT TYPE 1-5 AGE 6-7;
   IF TYPE='TEEN' & AGE>19 THEN DO;
      FILE LOG;
      PUT 'TYPE AND AGE DON''T MATCH ' AGE=;
      _ERROR_=1;
      END;
   CARDS;
ADULT 30
TEEN  25
TEEN  14
;
```

Output 4.1 ERROR Statement: Setting Current File to Log and Printing Message

```
1          SAS(R) LOG   OS SAS 5.XX           MVS/XA JOB ERROR    STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB ERROR HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1              DATA CHECK;
2                 INPUT TYPE $ AGE;
3                 FILE OUT;
4                 PUT TYPE 1-5 AGE 6-7;
5                 IF TYPE='TEEN' & AGE>19 THEN
6                    ERROR 'TYPE AND AGE DON''T MATCH ' AGE=;
7                 CARDS;


NOTE: FILE OUT(E1) IS:
      DSNAME=XXXXXX.XXXX.XXXXXX(XX),
      UNIT=DISK,VOL=SER=XXXXXX,DISP=OLD,
      DCB=(BLKSIZE=141,LRECL=137,RECFM=VBA)
TYPE AND AGE DON'T MATCH AGE=25

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

9             TEEN  25
TYPE=TEEN AGE=25 _ERROR_=1 _N_=2
NOTE: 3 LINES WERE WRITTEN TO FILE XXX(XX).
      THE MINIMUM LINE LENGTH IS 8.
      THE MAXIMUM LINE LENGTH IS 8.
NOTE: DATA SET WORK.CHECK HAS 3 OBSERVATIONS AND 2 VARIABLES. 953 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.99 SECONDS AND 376K.

11            ;
NOTE: SAS USED 376K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

FILE Statement

Operating systems: All

Introduction

The FILE statement specifies the current output file. The current output file is an external file specified by the most recently executed FILE statement. PUT statements write to the current output file or to the SAS log if no FILE statement is specified.

FILE statement options allow you to control how the output file is written. You can use FILE statement options to

- define variables that keep track of the current line and column pointer location
- specify headings to be printed at the beginning of each new output page
- control what happens when a PUT statement attempts to write beyond the current record length.

More than one FILE statement can be used in a DATA step. The FILE statement is executable and therefore can be used in conditional (IF-THEN) processing.

When you use the FILE statement, you may also need to use other related statements, such as PUT, RETURN, and FILENAME. The PUT statement builds and directs output lines to the external file named in the most recently executed FILE statement. An understanding of the RETURN statement is required if you use the HEADER= option with the FILE statement. PUT and RETURN statements are described later in this chapter. You may also need to understand how the FILENAME statement is used with the FILE statement to define external files. The FILENAME statement is described in the chapter "SAS Statements Used Anywhere."[1]

Types of files the SAS System can write Generally, the SAS System can write the same types of files that it can read. (See the INFILE statement description for the types of files the SAS System can read.)

Specifications

The form of the FILE statement is

FILE *fileref[(membername)] [typeoption options]*;

where

fileref

refers to the external file to which PUT statements write output lines. In most cases you must associate *fileref* with the name of this external file; that is, you must define the file to which output lines are written. Methods for defining files are described in the appendix

"Operating System Notes."

The following three special filerefs refer to output files that are automatically defined when you invoke the SAS System:

PRINT
: writes the lines produced by PUT statements on the standard SAS print file along with output produced by SAS procedures. When PRINT is the fileref, the SAS System uses carriage control characters and writes the lines with characteristics given below in the section entitled **Print files and non-print files.**

  Operating systems: All

PUNCH
: punches the lines produced by PUT statements on the standard SAS punch file (which is usually directed to cards). Use this fileref in operating environments that support card or card-image output.

  Operating systems: CMS, OS, VM/PC, and VSE

LOG
: writes the lines produced by PUT statements on the SAS log. Since output lines are by default written to the SAS log, a FILE LOG statement is needed only to restore the default action or to specify additional FILE statement options.

  At the beginning of each execution of a DATA step, SAS sets the fileref to LOG. Thus, the first PUT statement in the following DATA step always writes a line on the SAS log; the second PUT statement writes a line to the PRINT file:

```
DATA _NULL_;
   first PUT statement
   FILE PRINT;
   second PUT statement
more SAS statements
```

  Operating systems: All

(membername)
: identifies a member of an OS partitioned data set (PDS) to which output is directed. You must specify a member name after the fileref when you are writing to a member of a partitioned data set (PDS) and you do not specify the member in the control language that defines the file. When the current output file is a PDS member, a disposition of MOD (DISP=MOD) specified in the control language is not valid; nor can you specify the MOD option (described below).

  Using a member name in the FILE statement instead of in the control language for the job is useful when you want to store more than one member in the same partitioned data set in different DATA steps of the SAS job.

  Operating systems: OS

typeoption
: identifies the output file as a nonstandard external file that has special characteristics and requires a special access method. The *typeoption* **must** be specified immediately after the fileref. Valid *typeoptions* are as follows:

DA

indicates that you are updating records that were read using an INFILE statement with the DA option. The DA INFILE statement is used with an INPUT statement for

- key or sequential access to VSE DA (direct access) files (CKD disks only)
- random or sequential physical access to the blocks of a VSE file on an FBA disk
- random or sequential physical access to the blocks of a VSE or DA file on an CKD disk.

Similarly, the DA option is used to update a VSE disk file. The block must first be read using DA INFILE and INPUT statements. See the INFILE statement description for details on reading such files.

To update one of the above files, the fileref in the FILE statement **must** match the fileref in the DA INFILE statement. The keyword DA must immediately follow the fileref, and no other FILE statement options can be specified. Note that the SAS system option DAUPDATE must be in effect to use a DA FILE statement. The following statements illustrate these requirements:

```
OPTIONS DAUPDATE;
DATA;
   INFILE fileref1 DA;
   INPUT ... ;
   FILE fileref1 DA;
   PUT ... ;
```

The DA FILE statement can be used only for updating existing records. You cannot add new records or delete existing records. Key fields of VSE DA files cannot be altered.

Operating system: VSE

DLI

specifies that the output file is a DL/I data base. This typeoption is valid only if your site runs the SAS/IMS-DL/I software product.

Operating systems: OS, VSE

VSAM

specifies that the output file is a VSAM file. See the *SAS Guide to VSAM Processing* for more information on writing to VSAM files.

Operating systems: CMS, OS, VSE[2]

options

use the following options to control how lines are written to the current output file. Some of these options do not apply for all operating systems under which the SAS System runs. Use **Table 4.2** as a quick reference to options (and typeoptions) available on your operating system. A detailed description of each option follows the table.

Table 4.2 Options Available on Each Operating System

(Note: X indicates that the option is supported.)

Options	Operating Systems						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
BLKSIZE= | BLK= specifies the block size.		X	X		X		X
CLOSE= specifies the close disposition for the file.			X				
COLUMN= gives column location of the pointer on output line.	X	X	X	X	X	X	X
DA (typeoption) specifies physical direct access to a VSE file on disk.							X
DCB= specifies the fileref of file whose DCB information you want to use.		X	X		X		X
DEVTYPE= returns the device type information.			X				
DLI (typeoption) specifies DL/I data base as the output file.			X				X
DSCB= returns the DSCB of a non-VSAM disk data set.			X				
FLOWOVER | DROPOVER | STOPOVER determines action taken when pointer goes past the end of the line.		X	X		X		X
HEADER= labels statements to be executed when new page begins.	X	X	X	X	X	X	X
JFCB= returns the job file control block for the DD statement.			X				
LINE= gives the line location of the pointer on the output page.	X	X	X	X	X	X	X

continued on next page

Table 4.2 *continued*

Options	Operating Systems						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
LINESIZE= | LS= sets a maximum record length for the output file.	X	X	X	X	X	X	X
LINESLEFT= | LL= specifies the number of lines left on the output page.	X	X	X	X	X	X	X
LRECL= specifies the logical record length for the output file.	X	X	X		X	X	X
MOD writes output lines after existing lines in the output file.	X	X	X	X	X	X	
N= gives the number of output lines available to the pointer.	X	X	X	X	X	X	X
NOPRINT suppresses addition of carriage control characters.	X	X	X	X	X	X	X
NOTITLES | TITLES suppresses printing of title lines.	X	X	X	X	X	X	X
OLD writes output lines at beginning of file.	X	X	X	X	X	X	
PAGESIZE= | PS= specifies the number of lines per page.	X	X	X	X	X	X	X
PRINT writes a print file with carriage controls in column 1.	X	X	X	X	X	X	X
RECFM= specifies the record format.	X	X	X		X	X	X
UCBNAME= returns the unit name (device address) filed from the unit control block.			X				
VOLUME(S)= returns the volume serial.			X				
VSAM (typeoption) specifies that the output file is a VSAM file.		X	X				X

BLKSIZE=*value*
BLK=*value*
: specifies the block size of the output file. If BLKSIZE= is specified in the control language defining the file, it is not needed in the FILE statement. If BLKSIZE= values are given in both the control language and the FILE statement, the value in the FILE statement is used. If you do not specify a block size, SAS uses a default block size.

  The default block size for non-print files is the smaller of the value of the SAS system option FILEBLKSIZE(*device*) or the maximum block size for the output device, rounded down to a multiple of eighty. Under VSE, block size cannot be specified in the job control language.

  Operating systems: CMS, OS, VM/PC, and VSE

COLUMN=*variable*
: defines a SAS variable whose value is the current column location of the pointer. SAS automatically assigns the current column location to the COLUMN= variable.

  Operating systems: All

DCB=*fileref*
: gives the *fileref* of a file referenced in an earlier FILE or INFILE statement in the same DATA step. SAS uses that file's RECFM=, LRECL=, and BLKSIZE= information for the file specified in the current FILE statement. In other words, you can use the DCB= option instead of repeating RECFM=, LRECL= and BLKSIZE= options in a subsequent FILE statement.

  Operating systems: CMS, OS, VM/PC, and VSE

DROPOVER
: discards data items that exceed the output line length (as specified by the LINESIZE= option). If there is not enough space in the current line to write a data item, SAS drops (or ignores) the entire item. When this occurs, the column pointer remains positioned after the last value written in the current line. Thus, the PUT statement may write other items in the current output line if they fit in the space remaining or if the column pointer is repositioned. When a data item is dropped, the DATA step continues normal execution (_ERROR_=0). At the end of the DATA step a message is printed for each file from which data were lost.

  Use DROPOVER when you want the DATA step to continue execution if the PUT statement attempts to write past the current line length but you do not want the data item that exceeds the line length to be written on a new line.

  Operating systems: CMS, OS, VM/PC, and VSE

FLOWOVER
: causes data that exceed the current line length to be written on a new line. When a PUT statement attempts to write beyond the maximum allowed line length (as specified by the LINESIZE= option), the current line is written to the file and the data item that exceeds the current line length is written to a new line. If the PUT statement contains a trailing @, the pointer is positioned after the data item on the new line, and the next PUT statement writes to

that line. This process continues until the end of the input data is reached or until a PUT statement without a trailing @ causes the current line to be written to the file.

FLOWOVER is the default action for most types of files, except VSAM files. STOPOVER (described below) is the default for VSAM files. Under operating systems that do not support the FLOWOVER, DROPOVER, and STOPOVER options, SAS takes the action described for FLOWOVER.

Operating systems: CMS, OS, VM/PC, and VSE

HEADER=*label*

associates a label with a group of SAS statements that execute each time SAS begins a new output page. *Label* defines the first statement in the group. You must include a RETURN statement before the group and as the last statement in the group. Use the HEADER= option with print files.

The following DATA step illustrates the use of the HEADER= option:

```
DATA _NULL_;
   SET SPRINT;
   BY DEPT;
   FILE YEAR85 PRINT HEADER=NEWPAGE;
   PUT @22 DEPT @34 SALES;
   RETURN;
NEWPAGE:
   PUT @ 20 'SALES FOR 1985' /
         @25 DEPT=;
   RETURN;
```

The statements after the NEWPAGE label are executed when SAS begins printing a new page. You can use any SAS statement in the group of statements labeled for execution with the HEADER= option, even statements containing constants or variable names.

Operating systems: All

LINE=*variable*

defines a SAS variable whose value is the current relative line number within the group of lines specified by the N= option (described below). Thus, this variable can have a value from 1 up to the value specified by the N= option. If N= is not specified, the LINE= variable has a value of 1.

Operating systems: All

LINESIZE=*value*
LS=*value*

sets the maximum number of columns per line for reports and the maximum record length for data files. If PUT statements try to write a line that is longer than the value specified by the LINESIZE= option, the action taken by SAS depends upon whether the FLOWOVER, DROPOVER, or STOPOVER option is in effect. The default LINESIZE= value depends on the type of file. The following list gives default line sizes depending on how you specify the FILE statement:

FILE PUNCH	80
FILE LOG	LINESIZE= option*
FILE PRINT	LINESIZE= option*
FILE fileref PRINT	LINESIZE= option*
	or TLINESIZE= option*
FILE fileref (non-print)	LRECL=value[3]
FILE fileref VSAM	VSAM RECSIZE

*These are system options described in "SAS System Options."

SAS will not write a record that is longer than the LINESIZE= value. If you specify this option, the value must fall within the range allowed by the operating system.[4]

Operating systems: All

LINESLEFT=*variable*
LL=*variable*

defines a SAS variable whose value is the number of lines left on the current page. Consider the following DATA step:

```
DATA _NULL_;
   SET INFO;
   FILE PRINT LINESLEFT=L;
   PUT @10 NAME /
         @10 ADDRESS1 / @10 ADDRESS2 /
         @20 PHONE //;
   IF L<7 THEN PUT _PAGE_ @;
```

In this example, if there are fewer than seven lines left on the page, the PUT _PAGE_ @; statement begins a new page and positions the line pointer at line 1.

You can use the PAGESIZE= option (described below) to specify the number of lines per page.

Operating systems: All

LRECL=*value*

specifies the logical record length of the output file. If LRECL= is specified in the control language, it is not needed in the FILE statement. If LRECL= values are specified in both the control language and the FILE statement, the value in the FILE statement is used. If you do not specify a logical record length, SAS uses a default value.[5]

Operating systems: AOS/VS, CMS, OS, VM/PC, VMS, and VSE

MOD

writes the output lines after any existing lines on the file. The MOD option overrides the disposition specified in the control language.[6]

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, and VMS

N=PAGESIZE
N=PS
N=*value*

gives the number of output lines available to the pointer. The value of the N= option can be a number or the keyword PAGESIZE (PS). If the N= option is not specified and no # pointer controls are used in the current DATA step, one line is available; that is, the default value of N= is 1. If N= is not specified and you do use a # pointer control, N= has the highest value specified for a # pointer control in any PUT statement in the current DATA step. (The PUT statement description gives a complete discussion of the # pointer control in **Pointer Controls and Format Modifiers**.)

When the value of N= is 1, the SAS System writes each line before it begins the next. When the value of N= is 3, for example, lines are available to the pointer in groups of 3. Initially, lines 1, 2, and 3 are available, and you can move the pointer from line 1 to line 3 and back to line 1; lines 1, 2, and 3 are available until you move the pointer to line 4. Then lines 4, 5, and 6 are available, and so on. The pointer advances to line 1 of a new page when the PAGESIZE= value is reached or when a PUT _PAGE_ statement is encountered.

If the current output file is a print file, N= must have a value of either 1 or PAGESIZE.

Specify N=PS if you want to arrange each page of output before writing it. The following job produces a four-column telephone book listing; each column contains a name and a phone number:

```
DATA _NULL_;
   FILE PRINT N=PS;
   DO C=1, 30, 60, 90;
      DO L=1 TO 50;
         SET PHONE;
         PUT #L @C NAME $20. +1 PHONE $8;
         END;
      END;
   PUT _PAGE_;
```

The N=PS option makes all lines on the page available to the pointer. The L and C variables mark the current line and column of the pointer. The SET statement reads a SAS data set containing the names and telephone numbers. The PUT statement writes the NAME and PHONE values on the current line (the L value) at the current column (the C value). L's value is incremented by one until fifty names are written.

When the inner DO loop is satisfied, C is incremented to thirty to move the pointer over to the second column, and fifty more names are written in that column. When the outer DO loop is satisfied, the report includes four columns of fifty names each.

When the last value in the last column has been written, the PUT _PAGE_; statement writes the entire page. In the next execution of the DATA step, the C and L values begin at one again.

Operating systems: All

NOPRINT
: suppresses addition of the carriage control characters to column 1 of the output lines for a print file. The most frequent use of the NOPRINT option is in printing a file that already contains these carriage control characters in column 1.

  Operating systems: All

NOTITLES
NOTITLE
: suppresses printing of the current TITLE lines on the pages of print files (those where the fileref is PRINT, where the PRINT option appears in the FILE statement, or where the corresponding control language includes a SYSOUT= or "print" specification). When NOTITLES is omitted, the SAS System prints any titles currently defined. See "SAS Statements Used Anywhere" for a complete description of the TITLE statement.

  Operating systems: All

OLD
: writes the output lines at the beginning of the file. Using OLD is necessary only if you used MOD for the file in an earlier FILE statement and want to return to OLD, the default value.

  Operating systems: AOS/VS, CMS, PRIMOS, OS, VM/PC, and VMS

PAGESIZE=*value*
PS=*value*
: sets the number of lines per page for your reports.

  The value may range from 20 to 500. If no value is specified, the value of the system option PAGESIZE= is used. See "SAS System Options" for more information.

  If any TITLE statements are currently defined, the lines they occupy are included in counting the number of lines for each page.

  Operating systems: All

PRINT
: produces a printed report in which carriage control characters appear in column 1 of the output lines. The PRINT option is not necessary if you are using fileref PRINT.[7]

  Operating systems: All

RECFM=*recordformat*
: specifies the record format of the output file.[8]

  Operating systems: AOS/VS, CMS, OS, VM/PC, VMS, and VSE

STOPOVER
: stops processing the DATA step immediately if a PUT statement attempts to write a data item that exceeds the current line length. When this occurs, SAS discards the errant data item, writes the portion of the line built before the error occurred, and issues an error message. STOPOVER is the default for VSAM files.

  Operating systems: CMS, OS, VM/PC, and VSE

The following options, described with the INFILE statement, can also be used in the FILE statement on the operating systems listed with each.

CLOSE=	Operating system: OS
DEVTYPE=	Operating system: OS
DSCB=	Operating system: OS
JFCB=	Operating system: OS
UCBNAME=	Operating system: OS
VOLUME(S)=	Operating system: OS

When multiple FILE statements specify the same output file If more than one FILE statement specifies different options for the same output file, the options execute as though you specified all of them in one FILE statement. You do not have to list previously specified options in subsequent FILE statements for the same output file. If you specify an option more than once using different values, the last value that you specify is used. For example:

```
DATA _NULL_;
   INPUT NAME $ GIFT $ COST;
   FILE SHOPLIST LINESIZE=50;
   PUT NAME GIFT COST;
   FILE LOG;
   PUT COST;
   FILE SHOPLIST LINESIZE=20 PRINT;
```

In this program the first FILE statement specifies fileref SHOPLIST and the LINESIZE= option to create an output file with a line length of 50 columns. The second FILE statement writes all COST values on the SAS log. The third FILE statement again specifies SHOPLIST with an additional option and a new value for the LINESIZE= option. When this program executes, the output file referenced by fileref SHOPLIST contains carriage control characters because the PRINT option is specified and has a line length of 20 columns.

Print files and non-print files If the file that you specify in the FILE statement is to be printed, it must contain carriage control characters in column 1 of each line. A file that contains carriage control characters in column 1 is called a *print* file. The SAS System automatically produces print files when PRINT is specified as the fileref or when the PRINT option is used in the FILE statement.[9]

The SAS System inserts the following standard printer control characters into print files:

blank	single spacing
0	double spacing
minus (-)	triple spacing
1	begins a new page
plus (+)	overprints the preceding line.

Files that do not contain printer control characters are called *non-print* files. Column 1 of records in non-print files contains the data written there by the program that produced the records.

In addition to the presence or absence of control characters, print and non-print files may have other distinguishing characteristics.[10]

Notes

1. **AOS/VS, PRIMOS, and VMS**: In these environments use the FILENAME statement or operating system commands to define the current output file.

2. **CMS, and OS**: You can write VSAM files without specifying the VSAM option.

3. **AOS/VS, PRIMOS, and VMS**: If LRECL= is not specified, the default is variable length records with a maximum line size of 32700.

4. **AOS/VS, PRIMOS, and VMS**: The minimum line size is 10, and the maximum line size is 32700.

   CMS, OS, VM/PC, and VSE: The minimum line size is 10, and the maximum line size is 32756.

5. **AOS/VS**: You must specify LRECL= if the record format is F or FX (see the RECFM= option below). You can specify LRECL= in the FILE statement only.

   CMS, OS, VM/PC, and VSE: The default record length for non-print files is 80 and for print files, 137. Under VSE, you can specify LRECL= in the FILE statement only.

   VMS: You must specify LRECL= if the record format is F (see the RECFM= option below). You can specify LRECL= in the FILE statement only.

6. **CMS**: The MOD option is not valid for VSAM files.

   OS: Do not use the MOD option when the current output file is a member of a partitioned data set (PDS). The MOD option is not valid for VSAM files.

7. **OS**: The PRINT option is not necessary if your control language defines the file with a print class (SYSOUT=A) or with print characteristics (RECFM=VBA or FBA).

   VSE: The PRINT option is not necessary if the logical unit for the file is ASSGNed to a printer.

8. **AOS/VS**: Values for the RECFM= option can be F or FX for fixed-length files, V or VR for variable-length files, and D or DS for data-sensitive files. If the value of RECFM= is F or FX, you must also specify the LRECL= option (see above).

   CMS, OS, VM/PC: Values allowed for the RECFM= option are F, FB, V, VB, U, and VBS. F stands for "Fixed length," V stands for "Variable length," B stands for "Blocked," and U stands for "Undefined." Thus, RECFM=VB specifies variable length, blocked records. An A or M value can be included in any of these; each can end in T. If the RECFM= parameter is given in the control language describing the file, it is not needed in the FILE statement. If RECFM= is specified in both the control language and the FILE statement, the value in the FILE statement is used. The default record format value for non-print files is FB.

   VMS: Values for RECFM= can be F for fixed-length and V for variable length files. If RECFM= is F, you must specify LRECL= (see above).

   VSE: Values allowed for the RECFM= option are F, FB, V, VB, and U. F stands for "Fixed length," V stands for "Variable length," B stands for "Blocked," and U stands for "Undefined." Thus, RECFM=VB specifies variable length, blocked records. You can specify RECFM= only as a FILE statement option, not in the job control language. If RECFM= is not specified, the default value is FB.

9. **OS**: The SAS System also produces a print file if the DD statement for the file specifies a print classification (SYSOUT=A) or print characteristics (RECFM= VBA or FBA).

   VSE: The SAS System also produces a print file when the logical unit for the file is assigned to a virtual or real printer.

10. **CMS, OS, VM/PC, and VSE**: Print files have the default DCB characteristics RECFM=VBA, LRECL=137, and BLKSIZE=141. Non-print files have the default characteristics RECFM=FB, LRECL=80, and BLKSIZE= the value of the SAS system option FILEBLKSIZE(*device*) appropriate for the output device on which the file is being written.

FORMAT Statement

Operating systems: All

You can use the FORMAT statement to associate formats with variables in a DATA step. The formats can be either SAS formats or formats you have defined with PROC FORMAT. You can give the same format to several variables or different formats to different variables in a single FORMAT statement. When the SAS System prints values of the variables, it uses the associated format to print the values.

Using a FORMAT statement in the DATA step permanently associates a format with a variable; see the FORMAT statement in "Statements Used in the PROC Step" for information on assigning a format for the current PROC step only.

You can associate formats with variables with the statement

FORMAT *variables* [format]...;

These terms are included in the FORMAT statement:

variables names the variable or variables you want to associate with a format.

format gives the format you want SAS to use for writing values of the variable or variables in the previous variable list. Every format name ends with a period (for example, SEXFMT.) or has a period between the width value and number of decimal places (for example, DOLLAR8.2).

See "SAS Informats and Formats" for more information on the types of SAS formats that are available. See the FORMAT procedure for information on how to define your own formats.

If a variable appears in more than one FORMAT statement, the format given in the last FORMAT statement is used. A FORMAT statement used in a DATA step to associate one format with a variable can be overridden by a FORMAT statement in a later PROC step. The original format name remains stored with the variable in the data set. To disassociate a format from a variable, use the variable's name in a FORMAT statement with no format.

You can also assign formats with the ATTRIB statement. You can change a format assigned in a FORMAT statement with an ATTRIB statement and vice versa.

If you have used the FORMAT procedure to define your own formats, you need a FORMAT statement to associate the format with one or more variables:

```
PROC FORMAT;
   VALUE SEXFMT 1='MALE'
                2='FEMALE';
DATA ALL;
   INPUT NAME $ SEX @@;
   FORMAT SEX SEXFMT.;
   CARDS;
JANE  2  BILL  1
more data lines
;
```

The FORMAT procedure defines the SEXFMT. format. The FORMAT statement in the DATA step associates SEXFMT. with the variable SEX. When the values

of SEX are printed by any procedure, MALE and FEMALE are printed instead of the numbers 1 and 2.

When a FORMAT statement associating a variable or variables with a format is used in a DATA step, SAS associates the specified format with the variables in the SAS data sets being created and uses the format for printing values of the variables. The format name is stored with the data set. (Note: for permanently stored SAS data sets, if you associate a user-defined format with a variable, the format must be accessible when the data set is referenced, even if you will not be printing the variable's values. See the FORMAT procedure description for more information.)

When a variable that has been associated with a format in a FORMAT statement later appears in a PUT statement without a format specification, the variable's values are left aligned in the output field with leading blanks trimmed.

Using FORMAT for Datetime Values

If a variable contains SAS date, time, or datetime values, you must assign the variable a corresponding date, time, or datetime format in a FORMAT statement in order for the values to be printed in an understandable form. For example, consider this DATA step:

```
DATA INVENTRY;
   INPUT DESCRIPT $ ACQUIRED DATE7.;
   FORMAT ACQUIRED WORDDATE.;
   CARDS;
CABINET 15JAN84
DESK 03MAR85
more data lines
;
```

The FORMAT statement associates the SAS format WORDDATE. with the variable ACQUIRED, which contains SAS date values. (Note that the INPUT statement uses the SAS date informat DATE7. to read the data lines properly.) Without the FORMAT statement, values of ACQUIRED are printed as the number of days between January 1, 1960 and the ACQUIRED date, a large number that is difficult to interpret. See "SAS Informats and Formats" for more information on date, time, and datetime formats.

GO TO Statement

Operating systems: All

A GO TO (or GOTO) statement tells the SAS System to jump immediately to the statement indicated in the GO TO statement and begin executing statements from that point. The GO TO statement and destination must be in the same DATA step. The destination is identified by a statement label in the GO TO statement and the target statement.

The statement has the form:

GO TO *label*;
GOTO *label*;

where

label specifies a statement label that identifies the GO TO destination, which must be within the same DATA step. See **Labels, Statement** later in this chapter for more information.

The difference between the GO TO and LINK statements is in the action of a subsequent RETURN statement. A RETURN after a GO TO returns SAS execution to the beginning of the DATA step, whereas a RETURN after a LINK returns SAS execution to the statement following the LINK. In addition, a GO TO statement is often used without a RETURN statement; execution continues until another GO TO statement or the end of the DATA step, which contains an implied RETURN, is reached. A LINK statement is usually used with an explicit RETURN statement.

GO TO statements usually appear as the THEN clause in IF-THEN statements. In the example below, the SAS System jumps over the assignment statements to the OK label when X is between one and five inclusive:

```
DATA INFO;
   INPUT X Y;
   IF 1<=X<=5 THEN GO TO OK;
   X=3;
   COUNT+1;
   OK: SUMX+X;
   CARDS;
data lines
;
```

In the example above, the labeled statement is executed for every observation. Sometimes you want a labeled statement to be executed only under certain conditions. For example,

```
DATA RECORD;
   INPUT X Y Z;
   IF 1<=X<=5 THEN GO TO OK;
   X=3;
   COUNT+1;
   RETURN;
   OK: SUMX+X;
   CARDS;
data lines
;
```

The statement

```
SUMX+X;
```

is executed only for observations with X values between one and five inclusive. When the RETURN statement is executed, the SAS System outputs the current observation to data set RECORD and returns to the beginning of the DATA step and a new execution.

GO TO statements can often be replaced by DO-END statements. For example, using DO-END in the first example above results in:

```
DATA INFO;
   INPUT X Y;
   IF 1<=X<=5 THEN DO;
      X=3;
      COUNT+1;
      END;
   SUMX+X;
```

The second example above with DO-END and IF-THEN-ELSE is

```
DATA RECORD;
   INPUT X Y Z;
   IF 1<=X<=5 THEN DO;
      X=3;
      COUNT+1;
      END;
   ELSE SUMX+X;
```

See the discussion of the RETURN statement later in this chapter and "Data Step Applications" for other examples using the GO TO statement.

IF Statement

Operating systems: All

Introduction

In SAS processing there are two kinds of IF statements:

- conditional IF statements, written with a THEN clause, are used to execute a SAS statement only for observations that meet the condition specified in the IF clause. An optional ELSE statement gives an alternative action if the THEN clause is not executed.
- subsetting IF statements, without a THEN clause, are used to cause the SAS System to continue processing only those observations or records that meet the condition specified in the IF clause.

IF-THEN and IF-THEN/ELSE Statements

Use the IF-THEN statement when you want to execute a SAS statement for some but not all of the observations in the SAS data set being created. SAS evaluates the expression following the IF. When the expression is true for the observation being processed, the statement following THEN is executed. When the expression is false, SAS ignores the statement following THEN and executes the ELSE statement immediately following the IF. If no ELSE statement is present, SAS executes the next program statement.

The form of the IF-THEN statement is

IF *expression* **THEN** *statement*;

where

expression
is any valid SAS expression. SAS evaluates the expression in an IF statement to produce a result that is either non-zero, zero, or missing. A non-zero result causes the expression to be true; a result of zero or missing causes the expression to be false. For example, in the statement

```
IF X=3 THEN statement;
```

when the variable X has a value of 3, the result of evaluating the expression 3=3 is 1, that is, true. In the statement

```
IF X THEN statement;
```

the expression is true when variable X has a value other than 0 or missing. See "SAS Expressions" for more information on the way SAS evaluates expressions.

statement
can be any executable SAS statement or DO group. The following list identifies executable SAS statements: ABORT, assignment, CALL,

DELETE, DO, ERROR, FILE, GO TO, IF-THEN, INPUT, INFILE, LINK, LIST, LOSTCARD, MERGE, OUTPUT, PUT, null, sum, RETURN, SELECT, SET, STOP, and UPDATE.

For example, the statement

```
IF YEAR=1984 THEN COLOR='BLUE';
```

assigns the value 'BLUE' to a variable named COLOR when the value of YEAR is 1984. If the value of YEAR is not 1984, COLOR is not assigned a value; the value remains as it was before the IF statement.

In the example

```
IF YEAR<1980 THEN DELETE;
```

SAS deletes the observation when its value for YEAR is less than 1980.

The ELSE statement can immediately follow an IF-THEN statement to specify a statement that is to be executed when the condition of the IF is false.

The form of the ELSE statement is

ELSE *statement;*

where

statement
is any executable SAS statement or DO group as described above.

The statements

```
IF YEAR=1984 THEN COLOR='BLUE';
ELSE COLOR='RED';
```

assign 'RED' to the variable COLOR when the value of YEAR is not 1984.

Here are some other examples of IF-THEN/ELSE statements:

```
IF 0<AGE<1 THEN AGE=1;
IF RESPONSE=. THEN DELETE;
IF STATUS='OK' AND TYPE=3 THEN COUNT+1;
IF STATE='CA' OR STATE='OR'
   THEN REGION='PACIFIC COAST     ';
ELSE IF STATE='NC' OR STATE='VA' OR STATE='MD'
   THEN REGION='MID ATLANTIC COAST';
IF HOURS>40 THEN LINK OVERTIME;
ELSE IF 0<HOURS<=40 THEN LINK REGULAR;
ELSE PUT 'PROBLEM OBSERVATION' _ALL_;
```

To execute more than one statement for a true condition, follow THEN with a DO group:

```
IF ANSWER=9 THEN DO;
   ANSWER=.;
   PUT 'INVALID ANSWER FOR ' ID=;
   END;
ELSE ANSWER=ANSWER*10;
```

See the **DO Statement** description earlier in this chapter for more information.

IF-THEN/ELSE statements can be nested, as in this example:

```
IF X=0 THEN
   IF Y¬=0 THEN PUT 'X ZERO, Y NONZERO';
   ELSE PUT 'X ZERO, Y ZERO';
ELSE PUT 'X NONZERO';
```

The first ELSE statement pairs with the second IF statement and the last ELSE statement pairs with the first IF statement.

In the example above you can use the SELECT statement instead of nested IF-THEN/ELSE statements, as follows:

```
SELECT (X);
   WHEN (0) DO;
      SELECT (Y);
         WHEN (0) PUT 'X ZERO, Y ZERO';
         OTHERWISE PUT 'X ZERO, Y NONZERO';
         END;
      END;
   OTHERWISE PUT 'X NONZERO';
   END;
```

The **SELECT Statement** is described later in this chapter.

Subsetting IF Statement

Use the subsetting IF statement to cause the SAS System to continue processing only observations from the file or data set being used as input to a DATA step that meet the condition specified in the IF statement. The resulting SAS data set therefore contains a subset of the original observations.

The form of the subsetting IF is

IF *expression*;

where

expression
: is any valid SAS expression. The expression must conform to the rules in "SAS Expressions."

If the expression is true (is nonzero and nonmissing), SAS continues executing statements in the DATA step for the observation it is building. If the expression is false (0 or missing), SAS immediately returns to the beginning of the DATA step for another execution without outputting the observation. The remaining program statements in the DATA step are not executed.

For example, the statement

```
IF SEX='F';
```

results in a data set containing only observations with a SEX value of 'F'.

Here is an example that uses subsetting IF statements to produce new data sets that are subsets of an original data set.

```
DATA POPULACE;
   INPUT NAME $ SEX $ AGE MARITAL $;
   CARDS;
data lines
;
DATA SENIORS;
   SET POPULACE;
   IF AGE>=65;
DATA WIVES;
   SET POPULACE;
   IF SEX='F' AND MARITAL='M';
```

Data set SENIORS contains only those observations from POPULACE with an AGE value of 65 or greater. The data set WIVES contains those observations for which the SEX value is 'F' and the MARITAL value is 'M'.

INFILE Statement

Operating systems: All

Introduction

An INFILE statement identifies an external file (that is, a non-SAS file) that you want to read with an INPUT statement. The external file can be on disk, tape, cards, or can be data entered from a terminal. (SAS data sets are not read with an INFILE statement. Instead, use a SET, MERGE, or UPDATE statement to read a SAS data set in a DATA step.)

For example, suppose you want to analyze sales information in an external file with PROC TABULATE. Before invoking PROC TABULATE you need to create a SAS data set from the external file with the INFILE and INPUT statements.

INFILE statement options describe the input file's characteristics and specify how it is to be read with an INPUT statement. Using INFILE options you can:

- define variables whose values reflect the current pointer location, the length of the last line read, or whether the current line is the last in the file
- define what happens when the pointer reaches past the end of the current line
- restrict processing of the file by skipping lines at either the beginning or end of the file or both.

Note: important operating-system-specific information is given in footnote form at the end of this section. Be sure you read notes pertaining to your operating system.

Where the INFILE statement goes Since the INFILE statement identifies the file to be read, the INFILE statement must execute before the INPUT statement that reads the data lines.

You can read several external files within one DATA step. Each file must have a corresponding INFILE statement. The INPUT statement reads from the current input file, that is, the file pointed to by the most recently executed INFILE statement. The INFILE is CARDS (if there is a CARDS statement) or null until an INFILE statement executes.

When reading from more than one external file in a DATA step, it is possible to alternately access the files. For example, you can partially process one external file, go on to a different file, and return to the original file. An INFILE statement must be executed each time you want to access a file, even if you are returning to a file previously accessed.

Suppose your program reads from two files, referenced by the names EXFILE1 and EXFILE2. To access the files alternately, execute a new INFILE statement for each access:

```
DATA MYDATA;
   INFILE EXFILE1;
   INPUT ... ;
   more SAS statements
   INFILE EXFILE2;
   INPUT ... ;
   more SAS statements
   INFILE EXFILE1;
   INPUT ... ;
   more SAS statements
```

Notice that to read from EXFILE1 a second time, you must execute a second INFILE statement for EXFILE1.

When you use more than one INFILE statement for the same fileref, and you specify options in each INFILE statement, the effect is additive. That is, the options specified in each INFILE statement are added to the options specified in any subsequent INFILE statement(s) for that file.

Since the INFILE statement is executable, it can be used in conditional processing (in an IF/THEN statement, for example).

Types of files the SAS System can read You can use SAS software to read many kinds of external files. Table 4.3 shows which types of files can be accessed for each operating system. A few of the types in the table require special options (see the description of *typeoption* in **Specifications**, below). These files are called *nonstandard external files*. Files that do not require special options are called *standard external files*.

Table 4.3 File Types That the SAS System Can Read

File Organization	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
Sequential (disk)	s	s	s	s	s	s	s
Sequential (tape)	s	s	s	s		s	s
ISAM			s				
BDAM		s	s				s d
VSAM		s d	s d				s d
Partitioned		s	s		s		
Tapes	s	s	s	s		s	s

s = sequential access
d = direct access

Specifications

The form of the INFILE statement is

INFILE *fileref* | **CARDS** [(membername)] [*typeoption options*];

The specifications for the INFILE statement are described below:

fileref specifies the fileref (file reference) that identifies the input file.[1]

A *fileref* is a short name for the file you want to process. At some point prior to execution of the INFILE statement, you explicitly associate the fileref with the file's complete name. For some operating systems, this association is done with control language commands or statements. For other operating systems, the association is made by a SAS statement.[2]

The fileref **must** be specified and must come immediately after the keyword INFILE unless you are specifying CARDS (see below), using a CARDS statement, and reading in-stream data.

Operating systems: All

CARDS specifies that the input data immediately follow a CARDS statement in your SAS job. Use CARDS instead of a fileref when you want to use INFILE statement options to read in-stream data (on cards or entered at a terminal). In other words, use both an INFILE statement and a CARDS statement: substitute the word CARDS for the *fileref* in the INFILE statement:

```
DATA EXAM;
   INFILE CARDS options;
   INPUT...;
   other SAS statements
   CARDS;
data lines
;
```

Operating systems: All

(membername) identifies a specific member of a partitioned data set. The membername must be enclosed in parentheses. This is an alternative to identifying the member in the control language that defines the file. If the member is not specified in the INFILE statement, it must be specified in the control language.

Operating systems: CMS, OS, and VM/PC

typeoption indicates that the file is one of the nonstandard external files SAS can process. If you are processing one of the nonstandard files, the typeoption specification **must** immediately follow the fileref. Valid values for *typeoption* are

DA Specify DA to process a code DA file, or for physical access to a disk file. See **Options for DA Files**.

Operating system: VSE

POWER Specify POWER to access a POWER queue entry. See **Options for POWER Queues**.

Operating system: VSE

VSAM Specify VSAM to process VSAM files. See **Options for VSAM Files**.

Operating systems: CMS, OS, and VSE

VTOC Specify VTOC to access the Volume Table of Contents. See **Options for VTOC Files**.

Operating systems: OS and VSE

Note: your installation may support other typeoptions. See your installation representative for details.

options specifies one or more INFILE statement options. Options for standard external files are described in the next section. Options for nonstandard files are discussed in separate sections after the standard file options.

Table 4.5 at the end of this section is a summary of all INFILE statement options. It includes a brief description of each option and shows the operating

systems that support each option. This table is subdivided into sections according to whether the options are for standard or nonstandard files.

Options for Standard External Files

The options for nonstandard files (DA, POWER, VSAM, and VTOC) are discussed separately after this list of standard file options.

BLKSIZE=*blocksize*
: is the file block size. BLKSIZE= is required only if your input lines are in a non-labeled file.[3]

  You do not need to specify BLKSIZE= in both the control language and the INFILE statement. If you specify it in both places, the INFILE statement value supersedes. If you do not specify a blocksize in either place, the SAS System uses the blocksize from the file label, if there is one. If there is no file label, blocksize is the maximum permitted blocksize for the device.

  Operating systems: CMS, OS, VM/PC, and VSE.

BSAM
: uses the basic sequential access method (BSAM) to read physical blocks rather than logical records. This option is required if you use the CCHHR= option.

  Operating systems: OS

CCHHR=*variable*
: specifies a character variable to which the physical address (cylinder head record) of a record is returned. This option applies to files on CKD disks only, and it is used only if the BSAM option is also specified.

  The CCHHR= option also has a special use for DA and VTOC processing. See the separate sections on DA and VTOC for details.

  Operating system: OS

CLOSE=*keyword*
: indicates how a tape volume is to be positioned at the end of the DATA step. Values for *keyword* can be:[4]

REREAD	logical beginning of the data set
LEAVE	logical end of the data set
REWIND	physical beginning of volume
FREE	dynamically deallocate (MVS only)
DISP	as implied by the control language.

  Operating systems: OS and VSE

COLUMN=*variable*
COL=*variable*
: defines a variable that the SAS System sets to the column location of the input pointer. For example, these statements

```
DATA;
   INFILE A COLUMN=C;
   INPUT @5 X 3.;
   PUT C=;
```

produce the line

```
C=8
```

Operating systems: All

DCB=*fileref*
specifies the fileref of an input file used earlier in the same DATA step whose DCB attributes you want to use for the current input file.

Operating systems: OS and VSE

DEVTYPE=*variable*
defines a character variable (minimum length 24) that the SAS System sets to the device type from information in the operating system DEVTYPE macro.

You can use DEVTYPE= to access control block information both initially and at DD statement concatenation boundaries. The SAS System sets the EOV= variable (described below) to 1 when a concatenation switch occurs, indicating that the DEVTYPE= variable contains new information.

Operating systems: OS and VSE

DSCB=*variable*
defines a character variable (minimum length 96) that the SAS System sets to the Data Set Control Block (DSCB) information from a non-VSAM disk data set.

The DSCB= option lets you access control block information both initially and at concatenation boundaries. The SAS System sets the EOV= variable (described below) to 1 when a concatenation switch occurs, indicating that the DSCB= variable contains new information.

Operating systems: OS and VSE

END=*variable*
defines a variable that the SAS System sets to 1 when the current line is the last in the input file. The value of the END= variable is 0 until the last line is processed. You cannot use the END= option for UNBUFFERED files or for files allocated to your terminal, including a CARDS external file.

Operating systems: All

EOF=*label*
is a statement label that you specify as the object of an implicit GO TO when the INFILE statement reaches end-of-file. The SAS System jumps to the labeled statement when an INPUT statement attempts to read from a file that has no more records.

The EOF= option (and not END=) should be used when the DATA step uses

- multiple INPUT statements
- INPUT statements that read more than one data line at a time
- INPUT statements that execute within conditional constructs
- UNBUFFERED files, including a CARDS external file.

```
DATA;
   INFILE INPUT1 EOF=NEXT;
   INPUT...;
   RETURN;
```

```
NEXT: INFILE INPUT2 EOF=LAST;
   INPUT...;
   RETURN;
LAST: INFILE INPUT3;
   INPUT...;
```

The following is an explanation of why the example above shows the EOF= option used twice:

- In the INFILE INPUT1 statement, EOF=NEXT directs the SAS System to the label NEXT when all data have been read from the external file INPUT1.
- NEXT labels another INFILE statement naming the external file INPUT2. The EOF=LAST specification directs the SAS System to the LAST label when all data have been read from INPUT2.
- LAST labels another INFILE statement that names another external file, INPUT3.

Operating systems: All

EOV=*variable*
: defines a variable that the SAS System sets to 1 at the end of each DD statement in a series of concatenated files. That is, the EOV= variable is set to 1 when the last record in a file in a series of concatenated files is read. You must reset the EOV= variable back to 0 after the SAS System encounters each boundary.

  Operating system: OS

FIRSTOBS=*linenumber*
: indicates that you want to begin reading the input file at the line number specified, rather than beginning with the first record. For example, to start with record 100 you might use this INFILE statement:

```
INFILE D FIRSTOBS=100;
```

  Operating systems: All

FLOWOVER
: specifies the action to be taken if the INPUT statement reads past the end of the current record. When FLOWOVER is in effect, the INPUT statement reads the next record and continues reading data from the new record.

  Operating systems: All

JFCB=*variable*
: defines a character variable (of length 176 or greater) into which the SAS System puts job file control block (JFCB) information. The JFCB= option makes it possible to access OS control block information initially and at DD statement concatenation boundaries.

  The SAS System sets the EOV= variable (described earlier) to 1 when a concatenation switch occurs, indicating that the JFCB= variable contains new information.

  Operating system: OS

LENGTH= *variable*
: defines a variable that the SAS System sets to the length of the current INPUT line.[5]

You can reset the value of the LENGTH= variable in programming statements. This is useful when copying the input file to another file with PUT _INFILE_; you can use the LENGTH= option to truncate the copied records. For example, the statements below truncate the last 20 columns from the input lines before they are copied to the output file:

```
DATA;
   INFILE TAPE LENGTH=L;
   INPUT;
   L=L-20;
   FILE TAPEOUT;
   PUT _INFILE_;
```

Operating systems: All

LINE=*variable*
defines a variable that the SAS System sets to the line location of the INPUT pointer. The value of the LINE= variable is the current relative line number within the group of lines specified by the N= option (described below). Thus, the value of the LINE= variable ranges from 1 up to the value of the N= variable. Without the N= option, the LINE= variable has a value of 1, since by default only one line is available to the pointer. For example, the statements

```
DATA;
   INFILE B N=2 LINE=L;
   INPUT NAME 1-10 #2 ID 3-5;
   PUT L=;
```

produce the line

```
L=2    .
```

The LINE= option cannot be used with direct access, since by definition, direct access gets only one line at a time.

Operating systems: All

LINESIZE=*linesize*
LS=*linesize*
limits the record length available to the INPUT statement when you do not want to read the entire record.

For example, say that your data lines contain a sequence number in columns 73 through 80. You could use the INFILE statement

```
INFILE C LINESIZE=72;
```

to restrict the INPUT statement to the first 72 columns of the lines and prevent inadvertently reading a sequence number as data.

If an INPUT statement attempts to read past the column specified by LINESIZE=, the action taken depends on which of the FLOWOVER, MISSOVER, and STOPOVER options are in effect.

Operating systems: All

LRECL=*logical record length*
specifies the logical record length. Valid record sizes are from 1 to 32767.[6]

Operating systems: All

MISSOVER

prevents a SAS program from going to a new input line if it does not find values in the current line for all the INPUT statement variables. When an INPUT statement reaches the end of the current record, values that are expected but not found are set to missing.

For example, suppose you are reading temperature data. Each input line contains from 1 to 5 temperatures:

```
DATA WEATHER;
   INFILE CARDS MISSOVER;
   INPUT TEMP1-TEMP5;
   CARDS;
97.9 98.1 98.3
98.6 99.2 99.1 98.5 97.5
;
```

The SAS System reads the three values on the first data line as values of TEMP1, TEMP2, and TEMP3. The MISSOVER option causes the SAS System to set the values of TEMP4 and TEMP5 to missing for that observation because there are no values for those variables in the current input line.

When the MISSOVER option **is not used**, the SAS System goes to the second data line for the TEMP4 and TEMP5 values, printing the message

```
NOTE: SAS WENT TO A NEW LINE WHEN INPUT
STATEMENT REACHED PAST THE END OF A LINE.
```

The SAS System reads data line 3 the next time it executes the INPUT statement.

Operating systems: All

N=*number*

defines the number of lines you want available to the input pointer.

When the N= option is not used, the number of lines available to the pointer is the highest value following a # pointer control in any INPUT statement in the DATA step. When you do not use a # pointer control, N= has a default value of 1.

The only time the N= option is necessary is when you are reading a variable number of lines per observation without a # pointer control. The N= value affects only the number of lines that the pointer can access at a time; it has no effect on the number of lines an INPUT statement reads.

Operating systems: all

OBS=*line number*

gives the last line that you want to read from a sequential input file. This option is especially useful when you want to test your SAS program using just a few of the records in your file. For example, the statement below processes only the first 100 records in the file:

```
INFILE INB OBS= 100;
```

You can use the OBS= and the FIRSTOBS= options together to read records from the middle of your file. For example, the statement below begins processing with record 100 and ends with record 200:

```
INFILE IN2 FIRSTOBS= 100 OBS= 200;
```

Operating systems: All

RECFM=*record format*
: specifies the record format for a file if your input lines are in a nonlabeled file.[7]

Operating systems: AOS/VS, CMS, OS, VM/PC, VMS, and VSE

START=*variable*
: gives the first column number of the record that the PUT _INFILE_ statement is to write.

For example, say that you are making a copy of the file TAPE, but you do not want the first ten columns of the records copied. These statements copy only columns 11-80:

```
DATA;
   INFILE TAPE START=S;
   INPUT;
   S=11;
   FILE TAPEOUT;
   PUT _INFILE_;
```

Operating systems: All

STOPOVER
: stops processing the DATA step when an INPUT statement reaches the end of the current record without finding values for all variables in the statement.

When the STOPOVER option is specified and an input line does not contain the expected number of values, the SAS System sets _ERROR_ to 1, stops building the data set as though a STOP statement had executed, and prints the incomplete data line. Here is an example:

```
DATA;
   INFILE CARDS STOPOVER;
   INPUT X1-X4;
   CARDS;
1 2 3
5 6 7 8
9 4 0
;
```

When the SAS System reads the first data line, it does not find an X4 value. Because STOPOVER is specified in the INFILE statement, the SAS System sets _ERROR_ to 1, stops building the data set, and prints data line 1.

Without the STOPOVER option, the SAS System would print the message:

```
NOTE: SAS WENT TO A NEW LINE WHEN INPUT
STATEMENT REACHED PAST THE END OF A LINE
```

and continue to line 2 and read 5 as the value for X4. The next time the DATA step executes, the SAS System would read a new line, in this case, line 3.

Operating systems: All

UCBNAME=*variable*
: defines a character variable (minimum length 3) that the SAS System sets to the unit name (device address) from information in the OS unit control block (UCB).

  You can use UCBNAME= to access control block information both initially and at DD statement concatenation boundaries. The SAS System sets the EOV= variable to 1 when a concatenation switch occurs, indicating that the UCBNAME= variable contains new information.

  Operating system: OS

UNBUFFERED
UNBUF
: tells the SAS System not to perform the usual look-ahead read. When the UNBUFFERED option is specified, the SAS System never sets the END= variable to 1.[8]

  Operating systems: All

VOLUMES=*variable*
VOLUME=*variable*
: defines a character variable (minimum length 6) that the SAS System sets to the disk volume serial.[9]

  You can access control block information both initially and at concatenation boundaries. The SAS System sets the EOV= variable to 1 when a concatenation switch occurs, indicating that the VOLUME= variable contains new information.[10]

  Operating systems: OS and VSE

Options for DA Files

Operating system: VSE

Specifying DA in the *typeoption* position of the INFILE statement allows

- access to VSE DA (direct access) files, or
- access to VSE SD (sequential disk) files, or
- physical access to blocks on an FBA or CKD disk, within the extents of a VSE DA or VSE SD file.

Reading direct access files The DLBL JCL statement includes a code parameter that describes a file's type. The DA code is used for VSE direct access files, which are stored on CKD disks. You can access a DA file in a SAS program by specifying DA for *typeoption* in the INFILE statement. Access to the file can be sequential or direct (by key), depending on the DA INFILE statement options you specify. For sequential access, specify SEARCH=SEQ. To use direct access, specify a *search variable* with the KEY= option and SEARCH=KEY.

For each DA file record accessed, SAS can return several values:

- the feedback code
- the record's physical address (CCHHR)
- the relative track and record within the file.

The above values are returned if you specify *return variables* with the special INFILE options FEEDBACK=, CCHHR=, and TTR=. These and other options used for DA files are described later in this section.

Physical access to VSE disk files By specifying the INFILE statement DA option, you can have physical access (access by address) to files on FBA and CKD disks, whether the files are direct access (code DA) or sequential (code SD).

The physical blocks on an FBA disk file (512 bytes) can be retrieved

- sequentially
- by absolute block number
- by relative block number.

When retrieving by absolute or relative block number you must specify a *search variable*.

The physical blocks of a CKD disk file can be retrieved

- sequentially
- by relative track and record number within the file
- by address (CCHHR)
- by key.

When retrieving by relative track number, address, or key, you must specify a *search variable*.

An FBA or CKD block can contain more than one file record. Your SAS program can be used to segment a block into logical records, if required.

Return variables can be specified with the special INFILE options CCHHR=, TTR=, ABSBLOCK=, RELBLOCK=, and FEEDBACK=, so that you can capture values returned after each access. These and other options for physical access are described below.

For physical access to a disk file, the DLBL JCL statement for the file **must** specify DA in the code field, as in:

```
// DLBL filename,'file-ID',,DA
```

The DLBL statement must specify DA, even if the file being accessed is sequential (code SD). If you specify the DA option in the INFILE statement but you do not specify DA in the DLBL JCL statement, the file is not opened and the following error message is issued:

```
4661I INVALID DLBL FUNCTION
```

INFILE options for DA files and physical access to VSE disks The form of the DA INFILE statement is

INFILE *fileref* **DA** [*special and standard options*];

The keyword DA is specified in the *typeoption* position of the INFILE statement (following the fileref) to indicate that a DA file will be read.

DA files and files read with physical access are considered nonstandard files that require special INFILE statement options. The only standard INFILE option that is valid in a DA INFILE statement is the LINESIZE= option, which is described earlier in this chapter with the other standard INFILE options.

The special options for the DA INFILE statement are described below. A section of DA INFILE statement examples follows the option descriptions.

ABSBLOCK=*variable*
: specifies a numeric variable whose value is the actual number of a block within the file. This option applies to FBA disks only.

  The ABSBLOCK= variable can be used as a search variable or a return variable.

  - You can access specific blocks within the file by searching the file on the basis of the value of the ABSBLOCK= variable. The ABSBLOCK= variable is the *search variable* if you specify SEARCH=ABSBLOCK in the INFILE statement (see the description of SEARCH= below).

- Alternatively, you can access the file sequentially or by relative block number, and use the ABSBLOCK= variable to capture the actual block number of the block that is accessed. The ABSBLOCK= variable is a *return variable* for block information if SEARCH= **does not** specify ABSBLOCK.

If the ABSBLOCK= variable is the search variable and its value specifies a block that is not within the file's extents, the FEEDBACK= variable (below) is set to 4.

CCHHR=*variable*
specifies a character variable (5-byte minimum length) whose value is the physical address of a block. This option applies to CKD disks only.

The CCHHR= variable can be used as a search variable or a return variable.

- You can access specific addresses within the file by searching the file on the basis of the value of the CCHHR= variable. The CCHHR= variable is the *search variable* if you specify SEARCH=CCHHR in the INFILE statement (see the description of SEARCH= below).
- Alternatively, you can access the file sequentially or by relative track number, and use the CCHHR= variable to capture the address of the block that is accessed. The CCHHR= variable is a *return variable* for address information if SEARCH= **does not** specify CCHHR.

If the CCHHR= variable is the search variable, and its value specifies an address that is not within the file's extents, the FEEDBACK= variable (below) is set to 4.

FEEDBACK= *variable*
specifies a variable whose value is the return code for the most recent read request. The variable's value is 0 when a read request is successful. The value is set to 4 if the read request failed. The DATA step terminates at a failed read request unless the FEEDBACK= option is specified. When FEEDBACK= is specified, you must reset the FEEDBACK= variable to 0 in order to continue processing.

KEY=*variable*
specifies a variable whose value is the key associated with a particular block in the file. The KEY= variable's type and length must match the type and length of the key field. This option is for DA files on CKD disks only.

The KEY= variable can be used as a search variable or a return variable.

- You can access specific records within the file by searching the file on the basis of the value of the KEY= variable. The KEY= variable is the *search variable* if you specify SEARCH=KEY in the INFILE statement (see the description of SEARCH= below).
- Alternatively, you can access the file's records sequentially and use the KEY= variable to capture the key of the record that is accessed. The KEY= variable is a *return variable* for the key information if SEARCH= **does not** specify KEY.

If the KEY= variable is the search variable, and its value specifies a key that is not within the file's extents, the return code is set to 4.

If you specify the KEY= option, you must also specify the KEYLEN= option (see below). KEYLEN= must be specified before KEY= in the INFILE statement.

KEYLEN=*nnn*
KEYLTH=*nnn*
specifies the length of the KEY= variable. This value must match the key length defined when the file was created.

The KEYLEN= option must be specified before the KEY= option in the INFILE statement.

RELBLOCK=*variable*
specifies a numeric variable whose value is the relative number of a block within the file. (The first block of a file has relative block number zero.)

The RELBLOCK= variable can be used as a search variable or a return variable.

- You can access specific blocks within the file by searching the file on the basis of the value of the RELBLOCK= variable. The RELBLOCK= variable is the *search variable* if you specify SEARCH=RELBLOCK in the INFILE statement (see the description of SEARCH= below).
- Alternatively, you can access the file sequentially or by absolute block number, and use the RELBLOCK= variable to capture the relative block number of the block that is accessed. The RELBLOCK= variable is a *return variable* for block information if SEARCH= **does not** specify RELBLOCK=.

If the RELBLOCK= variable is the search variable, and the requested block is not within the file's extents, the FEEDBACK= variable is set to 4.

SEARCH=*method*
specifies how the file is to be read, that is, what the search criterion will be. The value of *method* can be ABSBLOCK, CCHHR, KEY, RELBLOCK, SEQ, or TTR. Table 4.4 provides details on each of these values. Only one method can be specified.

Table 4.4 Values for the SEARCH= Option

SEARCH= Value	Search Criterion	Disk Type	Comments
ABSBLOCK	absolute block number	FBA	for access by absolute block number. A search variable must be specified with the ABSBLOCK= option.
CCHHR	cylinder head record number	CKD	for access by physical address (CCHHR). A search variable must be specified with the CCHHR= option.

continued on next page

Table 4.4 *continued*

SEARCH= Value	Search Criterion	Disk Type	Comments
KEY	key value	CKD	for access by key. A search variable must be specified with the KEY= option. KEYLEN= must also be specified.
RELBLOCK	relative block number	FBA	for access by relative block number. A search variable must be specified with the REBLOCK= option.
SEQ	sequential	FBA CKD	for sequential access. Variables specified by other DA options are return variable(s).
TTR	relative track and record	CKD	for access by relative track and record within the file. A search variable must be specified with the TTR= option.

For all SEARCH= values except SEQ, you must specify the corresponding option naming a *search variable*. In other words, if you specify SEARCH= ABSBLOCK, you must also specify the ABSBLOCK= option. If you specify SEARCH=CCHHR, the CCHHR= option must also be specified, and so on. There can be only one search variable for a file.

When SEARCH=SEQ

- the blocks are retrieved sequentially, and
- any variables specified by ABSBLOCK=, CCHHR=, KEY=, RELBLOCK=, or TTR= are *return variables*. (That is, they contain values returned when blocks or records are retrieved.) There can be more than one return variable for a file.

You **must** specify either the SEARCH= option or one of the other options. If you do not specify one of these options, the DATA step terminates. This INFILE statement causes such an error:

```
INFILE FBAFILE DA;
```

(We recommend that you always specify SEARCH=. However, if you do not specify SEARCH= but you do specify ABSBLOCK=, CCHHR=, KEY=, RELBLOCK=, or TTR=, then the option that is specified first is used for searching.)

TTR=*variable*
: specifies a variable whose value is a relative track and record within the file. This option is for CKD disk files only. The TTR= variable

must be a character variable with a length of at least 3. The first block of a file has a TTR value of '000001'X (hex), that is, relative track zero, record one.

The TTR= variable can be used as a search variable or a return variable.

- You can access specific tracks within the file by searching the file on the basis of the value of the TTR= variable. The TTR= variable is the *search variable* if you specify SEARCH=TTR in the INFILE statement (see the description of SEARCH= below).
- Alternatively, you can access the file sequentially or by address, and use the TTR= variable to capture the relative number of the track that is accessed. The TTR= variable is a *return variable* for address information if SEARCH= **does not** specify TTR.

If the TTR= variable is the search variable, and its value specifies a track that is not within the file's extents, the FEEDBACK= variable is set to 4.

Examples of the DA INFILE Statement

Direct access to a DA file This example shows the use of the SEARCH=, KEY=, and KEYLEN= options to read a DA file using direct access. The file's fileref is BYKEY. The search variable, KEYVAR, is a variable in the SAS data set IDKEYS; its value indicates which record should be read. Notice that the FBVAR variable contains feedback values.

```
DATA _NULL_;
   SET IDKEYS;
   INFILE BYKEY DA SEARCH=KEY KEYLEN=11 KEY=KEYVAR FEEDBACK=FBVAR;
   INPUT @1 FIELD1 ... ;
```

Sequential access to a DA file DA files can also be accessed sequentially, as the next example shows. A KEY= variable is specified, but because SEARCH=SEQ, the KEY= variable is a return variable. (That is, the keys of the records that are read will be returned to the KEY= variable.)

```
INFILE SEQAC DA SEARCH=SEQ KEYLEN=20 KEY=RETKEY;
INPUT @1 FIELD1 ... ;
```

Physical access to a CKD file In this example, a CKD file with fileref DAINSEA is read using physical access. Both SEARCH=CCHHR and CCHHR=CHRVAR are specified; therefore, the file is searched by relative block numbers contained in the search variable, CHRVAR. TTR=RTRVAR is also specified. RTRVAR is a return variable and will contain the relative track and record number of the blocks accessed by the program.

```
INFILE DAINSEA DA SEARCH=CCHHR CCHHR=CHRVAR TTR=RTRVAR;
CHRVAR='010F000B05'X;
INPUT @1 FIELD1 ... ;
```

Sequential access to a CKD file The following example illustrates sequential access and return variables. The CKD file has a fileref of DAINSEQ. Since SEARCH=SEQ, the CCHHR= and TTR= variables are return variables.

```
INFILE DAINSEQ DA SEARCH=SEQ CCHHR=RETCHR TTR=RETRTR;
INPUT @1 FIELD1 ... ;
```

Access by relative block to an FBA file In this example, an FBA file with fileref DAINFBA is read using physical access. Both SEARCH=RELBLOCK and

RELBLOCK=RBVAR are specified; therefore, the file is searched by physical addresses contained in the search variable, which is RBVAR. ABSBLOCK=ABVAR is also specified. ABVAR is a return variable and will contain the absolute block number of the physical blocks accessed by the program.

```
INFILE DAINFBA DA SEARCH=RELBLOCK ABSBLOCK=ABVAR RELBLOCK=RBVAR;
RBVAR=5;    /* RELATIVE TO 0 */
INPUT @1 FIELD1 ... ;
```

Notice that the value of RBVAR is set to 5. This means that the sixth block of the file is read.

Sequential access to an FBA file This example illustrate sequential physical access and return variables. The FBA file has a fileref of DAFBA. Since SEARCH=SEQ, the ABSBLOCK= and RELBLOCK= variables are return variables. FEEDBACK= is also specified, so return codes are captured.

```
INFILE DAFBA DA SEARCH=SEQ ABSBLOCK=RETAB RELBLOCK=RETRB FEEDBACK=FETFB;
INPUT @1 FIELD1 ... ;
```

Options for Power Queues

Operating system: VSE

You can read VSE/POWER LST or PUN queue entries with the INFILE statement. The queue entry requested must have a disposition of D or K before the DATA step executes. When an entry with a disposition of D is read until end-of-file, the entry is deleted from the queue when the DATA step ends. When an entry with a disposition of K is read until end-of-file, the entry remains in the queue with its disposition changed to L when the DATA step ends. If the DATA step does not read until end-of-file, the entry remains in the queue with a disposition of K.

VSE/POWER queues can be accessed only if

- SPOOL=YES was specified in the POWER macro during VSE/POWER generation, and
- XECB support was specified during VSE generation.

Your VSE systems programmer can tell you if these conditions have been fulfilled.

The form of the POWER INFILE statement is

INFILE *jobname* **POWER** [*special and standard options*];

The specification after the keyword INFILE, which is typically a fileref, must be the job name for the queue entry you want to access. The keyword POWER is specified in the *typeoption* position of the INFILE statement (following the job name) to indicate that a POWER queue entry will be read. There can be only one POWER INFILE statement per DATA step.

POWER queues are considered nonstandard files and require special INFILE statement options. The only standard INFILE options that are allowed are END=, EOF=, FIRSTOBS=, and LINESIZE=. The special options for POWER queue access are described below:

CC=NO | YES specifies whether or not the carriage control character is to be returned as part of the record. The default value is NO.

When CC=NO, the record length of a LST queue entry is 132, and the record length of a PUN queue entry is 80.

When you specify CC=YES, the carriage control character is returned in the first byte of the record. The record length of a LST queue entry is 133, and the record length of a PUN queue entry is 81.

You can use the LINESIZE= option to change the length of the returned entry.

CLASS=class specifies the VSE/POWER output class of the entry. The default class is A.

JNUM=*number* specifies the entry's VSE/POWER job number. If JNUM= is **not** specified, the job number is the first entry with a matching job name in the queue. For example,

```
DATA ONE;
   INFILE VSEOUTP POWER LST JNUM=12345 CC=YES
   LINESIZE=133;
```

LST indicates that the entry is in the LST queue. LST is the default.

PUN indicates that the entry is in the PUN queue.

PWD=*password* specifies the password for the entry if a password was specified when created the entry was created. Note: with VSE/POWER Version 2, a password **must** be associated with the entry.

USERID=*userid* specifies the eight-byte user identifier, if one was specified when the entry was created. This option applies to VSE/POWER Version 2.

Options for VSAM Files

Operating systems: CMS, OS, and VSE

The INFILE statement can be used to access VSAM files. In this section you will find the descriptions of the special INFILE options for VSAM files. VSAM file processing is complex, and a complete discussion is beyond the scope of this book. Refer to the *SAS Guide to VSAM Processing* for an in-depth discussion of VSAM processing with SAS programs, including a VSAM mini-primer and detailed examples.

The form of the VSAM INFILE statement is

INFILE *fileref* **VSAM** [*special and standard options*]**;**

The keyword VSAM is specified in the *typeoption* position of the INFILE statement (following the fileref) to indicate that a VSAM file will be read. The VSAM typeoption is not required in all cases; however, it **must** be used if any of the following conditions are true:

1. You are reading a VSAM file under VSE.
2. You are executing SAS programs under TSO on OS/VS2 Release 1.x (SVS), and the VSAM data set was dynamically allocated with the ALLOCATE command, rather than defined by a DD statement in your TSO LOGON cataloged procedure.
3. You have bypassed the VSAM catalog to determine the volume location of the VSAM component or cluster, and have not coded the AMP=('AMORG') parameter in the OS JCL language defining the VSAM component or cluster.

A VSAM file is considered a nonstandard file and requires special INFILE statement options. Some of the standard INFILE statement options can also be specified for VSAM processing. The standard options that can be used are

COL=*variable*
END=*variable*
EOF=*variable*
FLOWOVER
LINE=*variable*
LINESIZE=*variable*
MISSOVER
N=*variable*
STOPOVER.

The following special options are also used in the INFILE statement to access VSAM files:

BACKWARD
BKWD
: tells the SAS System to read a VSAM data set backwards. BACKWARD is valid only when reading sequentially.

BUFND=*integer*
: is the number of data buffers to be used for a VSAM input file. The BUFND= option can also be specified in the control language defining the VSAM data set. If you do not use the BUFND= option, VSAM provides a default value.

BUFNI=*integer*
: is the number of index buffers to be used for a VSAM input file. The BUFNI= option can also be specified in the control language defining the VSAM data set. If you do not use the BUFNI= option, VSAM provides a default value. For sequential processing, one index buffer is usually sufficient.

CONTROLINTERVAL
CTLINTV
CNV
: indicates that you want to read physical VSAM control intervals rather than logical records. When you specify the CONTROLINTERVAL option for a password-protected VSAM file, you must give a control interval access or higher level password with the PASSWD= option. Control interval access is typically used only for diagnostic applications or for reading a VSAM catalog.

ERASE=*variable*
: defines a numeric SAS variable that you must set in order to erase a VSAM record.

  To erase a record, set the ERASE= variable to a value of **1** before a PUT statement for the file executes. When you set the ERASE= variable to a value of 0, the record is **updated** (instead of erased) when the PUT statement executes. This is the default action if ERASE= is not specified.

  After a record is erased, the ERASE= variable is automatically reset to 0. Therefore, you must set it to 1 again to erase another record. This prevents the inadvertent deletion of a series of records.

  This option is valid only for KSDS and RRDS records. There is a VSAM restriction that records **cannot** be erased from an ESDS.

ERRORABEND
: tells the SAS System to abend if VSAM encounters a logical error while accessing the file. **This option is for diagnostic purposes and should be used only under the direction of a SAS consultant or your SAS installation representative.** Do not confuse this option with the SAS System option ERRORABEND. The INFILE statement option applies only to that file.

  The abend occurs as soon as the SAS System encounters the VSAM logical error and therefore is not affected by the FEEDBACK= option.

  Note that open files are not closed because the abend occurs immediately upon detection of a VSAM logical error. Therefore, updates made to VSAM files during the current DATA step could be lost if the control interval has not yet been written back to disk. VSAM catalog information, such as the number of records in the file, may also be incorrect after the abend.[11]

FEEDBACK=*variable*
FDBK=*variable*
: defines a numeric variable that the SAS System sets to the VSAM logical error code when a logical error occurs. FEEDBACK= is similar to the _FDBK_ automatic variable but is more flexible and better suited to handling VSAM logical errors. When the SAS System sets the FEEDBACK= variable, you must set it back to 0 to continue processing after a logical error. For more information, see the *SAS Guide to VSAM Processing*.

GENKEY
: specifies generic key processing. When GENKEY is specified, SAS programs treat the KEY= variable as the leading portion of a record's key. VSAM retrieves the first record whose key matches the generic key. Use this option if you plan to retrieve a series of records that have the same leading key field.

KEY=*variable*
KEY=(*list of variables*)
: indicates that keyed direct access is to be used to retrieve records from a KSDS or an ESDS through an alternate index. KEY= defines one or more character variables that you set to the key of the record you want retrieved. You can construct a key up to 256 characters long by defining a list of character variables. The SAS System builds the key by concatenating the variables. The key is extended with blanks (on the right) to the full key length that was set when the file was defined.

KEYGE
: specifies that retrieval requests with the KEY= option are for any record whose key is equal to or greater than the key specified by KEY=. This approximate key retrieval is useful when the exact record key is not known.

KEYLEN=*variable*
: specifies a numeric SAS variable that, when used with GENKEY, specifies the length of the key to be compared to the keys in the file. That is, the variable's value is the number of generic key characters passed to VSAM.

  In addition, the SAS System sets the variable specified by KEYLEN= to the actual key length defined in the cluster before the

DATA step executes. Thus, KEYLEN can be used to read the file's keys without knowing the key length in advance. Assign the value of the KEYLEN= variable to a different variable if you also intend to set the KEYLEN= variable for generic key processing.

KEYPOS=*variable*
specifies a numeric variable that the SAS System sets to the position of the VSAM key field. This option allows keys to be read without knowing the key position in advance. The variable is set to the column number, not the offset (which is column number minus 1).

PASSWD=*password*
gives the appropriate password for a VSAM file that has password protection. The password is replaced with Xs on the SAS log. The appropriate password is

- a read (or higher-level) password for a file that you are reading only
- an update (or higher level) password for a file that you are updating or loading
- a control interval (or higher level) password to read a file's control intervals directly.

RBA=*variable*
defines a numeric variable that you set to the relative byte address (RBA) of the data record (or control interval) you want to read.

The RBA= option indicates that addressed direct access is to be used for record retrieval, and is appropriate for an ESDS or a KSDS. The RBA= option can also be used to access the control intervals in a RRDS if the CONTROLINTERVAL option is specified.

RC4STOP
stops the DATA step from executing when opening a VSAM file results in a return code of 4. The SAS System normally treats a return code of 4 from OPEN as recoverable, and continues processing.

RECORDS=*variable*
defines a variable that the SAS System sets to the number of logical records in the VSAM file you are reading.

RESET
indicates that the VSAM file is to be reset to empty (no records) when it is opened. The RESET option applies only to loading a VSAM file that has been defined with the VSAM option REUSE. Specify this option to use a VSAM file as a work file by re-loading it in the DATA step. This option cannot be used if the file has an alternate index.

RRN=*variable*
defines a variable that you set to the relative record number (RRN) of the VSAM record you want to read or write. This option indicates that keyed direct access is to be used for record retrieval and storage and is appropriate only for an RRDS.

SEQUENTIAL
SEQ
specifies sequential VSAM record retrieval when either the RBA= (for an ESDS) or the RRN= (for a RRDS) direct access option indicates direct record storage for the PUT statement.

The SEQUENTIAL option is necessary only when adding new records after sequentially reading existing records in an ESDS or a RRDS.

SKIP

indicates that skip sequential access is to be used for record retrieval. Skip sequential access finds an initial record with keyed direct access and then retrieves records from that point on with sequential access. An unchanged KEY= or RRN= value indicates that subsequent records are to be retrieved sequentially. The SKIP option can be used for a KSDS or a RRDS.

UPDATE=*variable*

tells the SAS System that not every record read is to be updated when you are reading and writing records from a VSAM file. The UPDATE= option defines a numeric SAS variable that you set when you want to retrieve a record with update access.

- When you set the UPDATE= variable to a value of **1** before an INPUT statement executes, the record is retrieved for update. This is the default if UPDATE= is **not** specified.
- When you set the UPDATE= variable to a value of **0** before the INPUT statement executes, the record is **not** retrieved for update.

When you have both an INFILE and a FILE statement pair that reference the same VSAM file, records are retrieved for update by default. When a record is retrieved for update, no other user, including you, can access that particular record or any other records in the same control interval, until you free the record by executing a PUT or another INPUT statement for that file. The UPDATE= option is used to avoid "user lockout" when only a few of many records read need updating.

Options for VTOC Files

Operating systems: OS and VSE

A direct-access disk's *volume table of contents* or *VTOC* can be read with the INFILE statement. The form of the VTOC INFILE statement is

INFILE *fileref* **VTOC** [*special and standard options*]**;**

The fileref specification differs between operating systems.[12] The keyword VTOC indicates that the VTOC file is to be read.

The VTOC is considered a nonstandard file and requires special INFILE statement options. In addition, all standard INFILE statement options can be specified for VTOC processing.

The following special options are used in the INFILE statement to access VTOC files:

CCHHR=*variable*

defines a character variable whose value is the cylinder-head-record address of each VTOC record accessed. The value returned to the CCHHR= variable is a hexadecimal value of length 5. Use the $HEX10. format to display the returned address.

Operating systems: OS and VSE

CVAF

causes the SAS System to use the Common VTOC Access Facility (CVAF) of the IBM program product Data Facility/Device Support (DF/DS) for indexed VTOCs. If the VTOC is not indexed or if your installation does not support DF/DS, this option has no effect.

When you use CVAF and a CCHHR= variable, values returned to the CCHHR= variable for Format Five data set control blocks

(DSCBs) are not valid because indexed VTOCs do not include Format Five DSCBs.

Operating system: OS

Notes

1. **VSE**: When VTOC is specified for *typeoption* in the INFILE statement, the name specified in the fileref position is either a volume serial or a logical unit. When POWER is specified, the name is the job name of the queue entry. The sections on the POWER and VTOC *typeoptions* discuss these specifications.
2. Different operating systems use different statements to assign reference names and describe the input file:

- **AOS/VS**: Use the SAS FILENAME statement or the AOS/VSCREATE /LINK command.
- **CMS**: Use the FILEDEF command.
- **OS (batch)**: Use a DD statement.
- **OS (TSO)**: Use the ALLOCATE command.
- **PRIMOS**: Use the SAS FILENAME statement.
- **VMS**: Use the SAS FILENAME statement or the VMS ASSIGN command.
- **VSE (batch)**: Use the DLBL, EXTENT, and ASSGN statements.
- **VSE (ICCF)**: Use the /FILE job entry statement.

See the appendix on **Operating System Notes** for details and examples for your operating system.

3. **VSE**: You must specify either the BLKSIZE= or the LRECL= option in the INFILE statement unless you are reading a nonstandard file.
4. **VSE**: Values for *keyword* can be either LEAVE or REWIND.
5. **CMS, OS, VM/PC, and VSE**: For variable-length records, the length of the 4-byte length descriptor (RDW) is not included in the length value. For undefined format (RECFM=U) records, the length value is the length of the current block.
6. **AOS/VS, PRIMOS, and VMS**: The maximum LRECL= value is 32700.

   CMS, OS, and VM/PC: You do not need to specify the LRECL= option in both the control language and the INFILE statement. If you specify it in both places, the INFILE statement value overrides. If you do not specify a LRECL= value, the value from the file label, if there is one, is used. If there is no file label, the default LRECL= value depends upon the values (explicit or default) for BLKSIZE= and RECFM=.

   VSE: You cannot specify the LRECL= option in the control language. If you do not specify a LRECL= value, the default depends upon the the values for BLKSIZE= and RECFM=. Except when reading nonstandard files, either the LRECL= or the BLKSIZE= option, or both, must be specified in the INFILE statement. You must specify both the BLKSIZE= and the LRECL= options if the RECFM=FB.
7. **AOS/VS**: Values for RECFM= can be

- F or FX: fixed length
- V or VR: variable length
- D or DS: data sensitive.

CMS, OS, VM/PC, and VSE: Values for RECFM= can be

- F: fixed length
- V: variable length
- FB: blocked fixed length
- VB: blocked variable length
- U: undefined.

CMS, OS, and VM/PC: The following can be added to the end of the values above:

- A: the first byte of each record is an ANSI printer control character
- S: the file contains spanned records (records which span more than one block)
- M: the first byte of each record is a machine printer control character
- D: the file contains variable-length ASCII tape records.

You do not need to specify the RECFM value in both the INFILE statement and the control language defining the input file. If it is specified in both places, the INFILE statement value supersedes. If it is not specified in either place, the value from the file label, if there is one, is used. The default RECFM= value is U.

VMS: Values for RECFM= can be

- F: fixed length
- V: variable length.

The RECFM=F and LRECL=*n* options can be used together to read variable length records as if they were fixed. The variable length records are padded with blanks or truncated to the length specified by the LRECL= option.

VSE: Except when reading a nonstandard file, RECFM= **must** be specified in the INFILE statement. Only F(B), V(B), and U can be specified.

8. **CMS, OS and VM/PC**: UNBUFFERED is automatically in effect for files allocated to the terminal.
9. **OS**: The SAS System sets the VOLUME= value from information in the job file control block (JFCB). The file can span up to five volumes, in which case the value of the VOLUME= variable is the concatenated volume serials.
10. **VSE**: The VOLUME= option is valid only for typeoption VTOC, and only one volume serial is returned.
11. **OS and CMS**: The abend is a user abend with a completion code that is the same as the VSAM logical error code.

    VSE: The abend is a program interruption for an execute exception with register one containing the VSAM logical error code.
12. **OS**: The fileref specification for VTOC access is the DDname associated with the VTOC in the control language for the job.

    VSE: The fileref should be a volume serial or a logical unit, not a filename. For example,

```
DATA ONE;
   INFILE VOLABC VTOC
   RECFM=F BLKSIZE=140
   LRECL=140 CCHHR=CCVAR;
      .
      .
      .

DATA TWO;
   INFILE SYS101
   RECFM=F BLKSIZE=140
   LRECL=140 VOLUME=VOLVAR;
      .
      .
      .
```

See the SAS Sample library for usage examples. The IBM manual, *VSE/Advanced Functions DASD Labels (SC24-5213)*, contains layouts of VTOC records.

Table 4.5 Summary of Standard and Special INFILE Statement Options

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
Standard External File Options							
BLKSIZE= specifies block size.		X	X		X		X
BSAM uses BSAM to read physical blocks of file.			X				
CCHHR= returns the cylinder-head-record address of each DSCB read.			X				
CLOSE= specifies the close disposition for the file.			X				X
COLUMN= gives column location of input pointer.	X	X	X	X	X	X	X
DCB= indicates that DCB from specified file be used for current file.			X				X
DEVTYPE= returns device type information.			X				X
DSCB= returns DSCB of a non-VSAM disk data set.			X				X
END= defines a variable indicating when current line is end-of-file.	X	X	X	X	X	X	X
EOF= specifies statement label for implicit GO TO at end-of-file.	X	X	X	X	X	X	X
EOV= indicates last file of concatenated files.			X				

continued on next page

Table 4.5 *continued*

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
FIRSTOBS= specifies first record to read.	X	X	X	X	X	X	X
FLOWOVER specifies action to take if INPUT reads past end of line.	X	X	X	X	X	X	X
JFCB= returns the job file control block for the file.			X				
LENGTH= specifies variable containing length of current line.	X	X	X	X	X	X	X
LINE= gives location of line pointer.	X	X	X	X	X	X	X
LINESIZE= limits record length read by INPUT statement.	X	X	X	X	X	X	X
LRECL= specifies record length of input records.	X	X	X	X	X	X	X
MISSOVER if values not present for variables in INPUT, assign missing values.	X	X	X	X	X	X	X
N= specifies number of lines available to pointer.	X	X	X	X	X	X	X
OBS= specifies last line to read from file.	X	X	X	X	X	X	X
RECFM= specifies the record format.	X	X	X		X	X	X
START= gives column number where PUT _INFILE_ starts writing.	X	X	X	X	X	X	X

continued on next page

Table 4.5 *continued*

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
STOPOVER stop DATA step if a record does not contain all values.	X	X	X	X	X	X	X
UCBNAME= returns device address field from unit control block.			X				
UNBUFFERED inhibits the standard look-ahead read.	X	X	X	X	X	X	X
VOLUMES= returns the volume serial.			X				X
DA File Options							
ABSBLOCK= specifies variable whose value is number of block.							X
CCHHR= specifies variable whose value is cylinder-head record address of block.							X
FEEDBACK= specifies variable containing return code for most recent read request.							X
KEY= specifies variable whose value is key for record.							X
KEYLEN= specifies length of KEY= variable.							X
RELBLOCK= specifies variable whose value is relative number of block.							X

continued on next page

Table 4.5 *continued*

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
SEARCH= specifies search criterion for reading file.							X
TTR= specified variable whose value is relative track and record number.							X
POWER File Options							
CC= specifies if carriage control character is returned with record.							X
CLASS= specifies output class of entry.							X
JNUM= specifies entry's job number.							X
PWD= specifies entry's password.							X
USERID= specifies user ID.							X
VSAM File Options							
BACKWARD reads VSAM file backwards.		X	X				X
BUFND= gives number of data buffers for VSAM file.		X	X				X
BUFNI= gives number of index buffers for VSAM file.		X	X				X

continued on next page

Table 4.5 *continued*

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
CONTROLINTERVAL reads physical VSAM control intervals.		X	X				X
ERASE= tells whether or not a VSAM record is to be erased.		X	X				X
ERRORABEND abend if logical error occurs while processing VSAM file.		X	X				X
FEEDBACK= is set to VSAM logical error code.		X	X				X
GENKEY specifies VSAM generic key processing.		X	X				X
KEY= specifies key of VSAM record to be read.		X	X				X
KEYGE requests first record with key equal to or greater than KEY= value.		X	X				X
KEYLEN= gives key length, and with GENKEY specifies generic key length.		X	X				X
KEYPOS= returns position of key field.		X	X				X
PASSWD= specifies password of protected file.		X	X				X
RBA= specifies relative byte address of record to be read.		X	X				X

continued on next page

Table 4.5 *continued*

Options	Operating System						
	AOS/VS	CMS	OS	PRIMOS	VM/PC	VMS	VSE
RC4STOP stops DATA step when file open return code is 4.		X	X				X
RECORDS= returns number of logical records in data component of VSAM cluster.		X	X				X
RESET specifies that file is to be reset to empty when opened.		X	X				X
RRN= specifies relative record number of record to be read.		X	X				X
SEQUENTIAL specifies sequential record retrieval from RRDS or ESDS when adding records.		X	X				X
SKIP use skip sequential access for retrieval.		X	X				X
UPDATE= specifies that not all records need to be retrieved with update access.		X	X				X
VTOC File Options							
CCHHR= returns the cylinder-head-record address.			X				X
CVAF allows compatible processing of OS/VS indexed VTOCs.			X				

INFORMAT Statement

Operating systems: All

The INFORMAT statement associates informats with variables. You can use it in a DATA step to specify a default informat for variables listed in an INPUT statement. You can also use it in some procedures such as PROC EDITOR and the FSEDIT procedure in the SAS/FSP software product.

You can associate informats with variables by using the statement

INFORMAT *variables* [*informat*] ...;

These terms are included in the INFORMAT statement:

variables
names the variable or variables you want to associate with an informat.

informat
gives the informat you want SAS to use for reading values of the variable or variables in the previous variable list. When an informat is specified in an INFORMAT statement, **only** the informat type (RB, IB, $, and so on) is used. Any width or decimal specification is ignored. Every informat name ends with a period (for example, $15.) or has a period between the width value and number of decimal places (for example, 8.2). All the informats described in "SAS Informats and Formats" except the $CHAR. and BZ. informats can be used. (You should not use the $CHAR. and BZ. informats because blanks are significant to them; however, list input mode uses blanks to indicate the end of an input field. Therefore, unexpected results may occur with embedded blanks in data lines when you use the $CHAR. and BZ. informats.)

When you use an INFORMAT statement to associate an informat with a variable and then use the variable's name without an informat in an INPUT statement, SAS reads the value using list input mode (described with the INPUT statement later in this chapter).

For example, these statements

```
INFORMAT FRSTNAME LASTNAME $15.;
INPUT FRSTNAME LASTNAME;
```

are equivalent to the statement

```
INPUT FRSTNAME : $15. LASTNAME : $15.;
```

The colon format modifier (:) is described with the INPUT statement later in this chapter.

INPUT Statement

Operating systems: All

Introduction

The INPUT statement describes the arrangement of values in an input record and assigns input values to corresponding SAS variables. A simple INPUT statement can read simple input data records. You may need to use the INPUT statement's advanced features to read more complicated input data.

The general form of the INPUT statement is

INPUT [*specification*]...;

There are four ways to describe a record's values in an INPUT statement:

1. column
2. list, or free-format
3. formatted
4. named.

The first three styles are the most common; named input can be used only with a particular form of input data. The following simple INPUT statements, reading the variables NAME (character) and AGE (numeric), illustrate the column, list, and formatted input styles. A character value is simply a sequence of characters. A dollar sign ($) following a variable in an INPUT statement indicates that it has a character value.

With **column input**, the column numbers containing a variable's value follow the INPUT statement variable:

```
INPUT NAME $ 1-8 AGE 11-12;
```

With **list input**, the variables to be assigned data values are simply listed in the INPUT statement:

```
INPUT NAME $ AGE;
```

With **formatted input**, you specify an *informat* after the INPUT statement variable. The informat indicates the variable's data type and field width.

```
INPUT NAME $CHAR11. AGE 2.;
```

Input styles can be mixed within an INPUT statement. For example, you can read NAME with column input and AGE with formatted input:

```
INPUT NAME $ 1-10 AGE 2.;
```

A DATA step can have one or many INPUT statements.

Kinds of data Standard-form data can be read with all four input styles. *Standard -form* data are stored with one digit or character per byte. One column on a terminal screen, printout, or punched card is equivalent to one byte. Data that are not in standard form can be read only with formatted input. Examples of non-standard form data are hexadecimal, packed decimal, and binary values. Both character and numeric data can be stored in standard or nonstandard forms.

A variable read with an INPUT statement is assumed to be numeric unless a dollar sign follows it in the INPUT statement, or the variable has been previously defined as character. If you specify a data type that is incompatible with the data value, SAS takes the action described in **How the SAS System Handles Invalid Data.**

Numeric data Standard numeric data can be represented in several ways: standard numeric, negative, fractions, hexadecimal, scientific notation (also called E-notation), packed decimal (signed or unsigned), binary (integer), floating point (real binary), zoned decimal (with or without legal blanks), blanks as zeroes, and special embedded characters ($, %, (), commas, and blanks). All of these numeric data forms must be read with formatted input except standard numeric and negative. For example, the standard numeric data value 23 can be expressed in any of the following ways:

DATA DESCRIPTION	DATA	RESULT
input right justified	23	23
input not justified	23	23
input left justified	23	23
input with leading zeros	00023	23
input with decimal point	23.0	23
in E−notation, $2.3x10^1$	2.3E1	23
in E−notation, $230x10^1$	230E−1	23
minus sign for negative numbers	−23	−23

Remember the following points when reading numeric data:

- a minus sign preceding the number (without an intervening blank) indicates a negative value.
- leading zeros and the placement of a value in the input field do not affect the value assigned to the variable.
- numeric data can have leading and trailing blanks, but cannot have embedded blanks (unless read with the COMMA. informat).
- to read decimal values from input lines that do not contain explicit decimal points, indicate where the decimal point belongs by a decimal parameter with column input, or an informat with formatted input. An explicit decimal point in the input data overrides any decimal specification in the INPUT statement.

The following are examples of input data that either must be read with an informat or that are invalid:

DATA	REASON
2 3	embedded blank requires COMMA. informat
− 23	embedded blank requires COMMA. informat
2,341	comma requires COMMA. informat
(23)	parentheses requires COMMA. informat
C4A2	hexadecimal value requires HEX. informat
1DEC84	date requires DATE. informat
23−	sign should precede the number
..	missing value is a single period
E23	not a number

See **Special Topics**, at the end of this chapter, for information on how the SAS System handles invalid numeric input.

Character data Input data can include any character. However, be careful when your input data include

- **leading blanks**. Character values beginning with blanks can cause

problems with list-style input because leading and trailing blanks are trimmed from a character value before the value is assigned to a variable. You can avoid problems when reading values with leading blanks by using formatted input with the $CHAR. informat. See the chapter "SAS Informats and Formats" for more information on informats.

- **semicolons** in data following a CARDS statement. See the description of the CARDS statement for more information and an example of how to handle semicolons in character data.

Column Input

Requirements With column input, the column numbers containing the value follow a variable in the INPUT statement. Column input can be used when the input values are

- in the same columns on all the input lines
- in standard numeric or character form.

For example, if the numeric variable COUNT occupies columns 7 and 8 in all lines of input data, this INPUT statement,

```
INPUT COUNT 7-8;
```

reads the value of the variable COUNT from columns 7 and 8 of each line of input data.

The form of the INPUT statement for reading one variable with column input is

INPUT *variable* [**$**] *startcolumn*
[*−endcolumn*] [*.decimals*];

where

variable
names the variable whose value the INPUT statement is to read.

$
indicates that the variable has a character, rather than a numeric, value. If the variable has been previously defined as character, the $ sign is not required.

startcolumn
is the first column of the input record that contains the variable's value.

−endcolumn
is the last column of the input record that contains the variable's value. If the variable's value occupies only one column, omit *endcolumn*.
For example,

```
INPUT NAME $ 1-10 PULSE 11-13 WAIST 14-15 AGE 16;
```

Values for the character variable NAME start in column 1 and end in column 10. Values for PULSE start in column 11 and end in column 13, and so forth. AGE occupies only column 16.

.decimals[1]
gives the number of digits to the right of the decimal if the input value does not contain an explicit decimal point. An explicit decimal point in the input value overrides a decimal specification in the INPUT statement. For example, the statement

```
INPUT NUMBER 10-15 .2;
```

reads the value of NUMBER with two decimal places. The following list shows some input data for the NUMBER variable and their values when SAS reads the data with the .2 specification.

INPUT	RESULTS
2314	23.14
2	.02
400	4.00
−140	−1.40
12.234	12.234 (input decimal overrides)
12.2	12.2 (input decimal overrides)

Features Features of column input are:

1. Input values can be read in any order, regardless of their position in the record. For example, the statement

   ```
   INPUT FIRST 73-80 SECOND 10-12;
   ```

   first reads a value of the variable FIRST from columns 73 through 80, and then a value for the variable SECOND from columns 10 through 12.
2. Character values can have imbedded blanks.
3. Character values can be from 1 to 200 characters long.
4. Values or parts of values can be reread. For example, in,

   ```
   INPUT ID 10-15 GROUP 13;
   ```

   columns 10-15 contain an ID value; the third digit of ID, column 13, is a group number.

Missing values Blank fields and fields containing only a period (.) are interpreted as missing values.

Blanks Column input ignores both leading and trailing blanks within the field. Thus, if you use the INPUT statement

```
INPUT COUNT 7-9;
```

and the value for COUNT is in columns 7 and 8 and column 9 is blank, the value in columns 7 and 8 is used. The SAS language does not treat trailing blanks as zeros as some other languages do (for example, FORTRAN).

If numeric values contain blanks that represent zeros or if you want to retain leading and trailing blanks in character values, you must read the value with an informat. See the **Formatted Input** section for more information.

List Input

Requirements With list input, the SAS System scans the input line for values, rather than reading from specific columns. Consider list input when

- input values are separated from each other by at least one blank
- periods, rather than blanks, represent missing values

- character input values have a maximum length of eight bytes, unless given a longer length in an earlier LENGTH, ATTRIB, or INFORMAT statement.

With list input, the values for the variables are delimited by one or more blanks or by the end of the input record, whichever comes first. Unless explicitly specified elsewhere, the length of character values is 8 bytes by default. To read longer values, use formatted input; that is, specify an informat, use column input, or define a longer length in a LENGTH, ATTRIB, or INFORMAT statement.

Features List input may be the easiest style to use since you can simply list the variables to be assigned values in the INPUT statement. It is not necessary to know what columns the data values occupy in the input record. However this style has some restrictions on the order of variables and missing values.

The form of the INPUT statement for reading one variable with list input is

INPUT *variable* [**$**] [**&**] ;

where

variable
names the variable whose value the INPUT statement is to read.

$
indicates that the preceding variable contains a character, rather than a numeric, value. If the variable has been previously defined as character, the $ sign is not required. For example,

```
INPUT NAME $;
```

&
indicates that a character value may have one or more single embedded blanks. Because a blank normally indicates the end of a data value with list input; when you use the ampersand modifier, indicate the end of the value with at least two blanks. For example, the following INPUT statement can read the values listed for the variables NAME and AGE:

```
DATA ONE;
   INPUT NAME $ & AGE;
   CARDS;
J. Jones  20
J. R. J.  31
;
```

Order of variables When reading with list input, the order of the variables listed in the INPUT statement and their corresponding values in the input data must be the same.

You cannot read values selectively with simple list input, although you can ignore all values after a given point. For example, suppose each input line contains five values for the variables A, B, C, D, and E, and you want only values for A, B, and D. With simple list input you cannot skip C, but you can omit E since it is after D, the last value you want:

```
INPUT A B C D;
```

Missing values Missing input values must be represented with a single period (.) to be read with list input.

Blanks You cannot read values that contain leading, trailing, or embedded blanks with list input because blanks indicate the end of data values. (To read

input values with leading or trailing blanks, use either the column or the formatted input style.) To read character values containing single embedded blanks, use the ampersand (&) modifier described above.

Formatted Input

Requirements An INPUT statement reading *formatted input* lists a SAS informat after the variable. An *informat* gives the data type and field width of an input value. The informat specified must be a SAS or a user-defined informat. See the chapter "SAS Informats and Formats" for more information about informats.

Features Features of formatted input are that you can:

- read input values with contiguous blanks
- read data in virtually any form
- use grouped format lists
- use format modifiers.

Formatted input is often used with the pointer controls discussed later in **Advanced INPUT Statement Features** and with format modifiers. Note, however, that neither pointer controls nor format modifiers are required to use formatted input.

The form of the INPUT statement for reading one variable with simple formatted input is

INPUT *variable* [*formatmodifier*]
informat **;**

where

variable
is the variable that the INPUT statement is to read.

formatmodifier
modifies the way the informat reads the input value. There are two format modifiers: the colon (:) and ampersand (&). They are described in **Grouped Format Lists**, below.

informat
gives the informat to use when reading the input value. An informat always includes or ends with a period (.), for example, 3.2 and $CHAR4. See the "SAS Informats and Formats" chapter for complete descriptions of all SAS informats.

Informats for character values begin with a dollar sign ($).

Actual decimal points included in the input override decimal specifications in an informat.

Grouped format lists When input values are arranged in a pattern, they can be described with a grouped format list. A grouped format list consists of two lists, each enclosed in parentheses: the first names the variables to be read; the second gives their corresponding informats. You can write shorter INPUT statements with format lists because

- the format list is recycled until all variables have been read
- numbered variable names can be used in abbreviated form to avoid listing all the individual variables.

You can use as many format lists as necessary in an INPUT statement, but format lists cannot be nested. For example, if the values for the five variables SCORE1-SCORE5 are arranged four columns per value without intervening blanks, the following INPUT statement

```
INPUT (SCORE1-SCORE5) (4. 4. 4. 4. 4.);
```

reads their values. However, if you specify more variables than informats, the format list is reused to read the remaining variables. Therefore, this shorter format list accomplishes the same thing:

```
INPUT (SCORE1-SCORE5) (4.);
```

When all the values in the variable list have been read, the INPUT statement ignores any directions remaining in the informat list.

You can use commas between items in format lists to delineate separate informats. In this statement,

```
INPUT (A B) ($,5.);
```

the comma indicates that the $ specification is associated with A, and that the 5. informat is associated with B. Without the comma,

```
INPUT (A B) ($ 5.);
```

the SAS System interprets the statement with a single informat, $5., and associates it with both A and B.

The *n** modifier[2] specifies in format lists that the next format is to be repeated *n* times. For example, say that you want to read first a value of the variable NAME, followed by the five SCORE values:

```
INPUT (NAME SCORE1-SCORE5) ($10. 5*4.);
```

This INPUT statement first reads a value of the variable NAME from columns 1 through 10; then reads the five SCORE values from the next 20 columns.

Format modifiers Format modifiers change the way an informat reads an input value. Specify a format modifier before the informat to which it applies. There are two format modifiers:

:

combines informats with the scanning feature of list input. The colon indicates that the value is to be read from the next nonblank column until

- the next blank column
- the length of the variable as previously defined has been read
- the end of the data line

whichever comes first. If the length of the variable has not been previously defined, its value is read and stored with the informat length.

For example, say that the first value on each input line is a last name that can be up to 15 characters long. Use the colon (:) format modifier before the $15. informat to indicate that the value ends at the first blank column, the end of the data line, or the end of 15 columns, **whichever comes first**. For example, the following INPUT statement

```
INPUT LASTNAME :$15.;
```

reads the following input data:

```
Smith 123 Highway
Longlastname 527 Avenue
```

Operating systems: All.

&

indicates that a character input value may contain one or more single embedded blanks and is to be read from the next nonblank column until one of the following is encountered:

- two consecutive blanks
- the length of the variable as first defined in the DATA step
- the end of the input line

whichever comes first.

For example, the statement

```
INPUT LASTNAME $ STATE & $;
```

indicates that the input line contains first a value for the character variable LASTNAME, and then a value, which can include one or more single blanks, for the character variable STATE.

Note: because the ampersand format modifier indicates that the input value may contain single embedded blanks, signify the end of the value with two blanks.

The & modifier has the same effect as the : modifier except that the terminating condition is two blanks rather than one. For example,

```
DATA ONE;
   INPUT LASTNAME & $15. FIRSTNAME;
   CARDS;
LONGLASTNAMEONE  JOHN
MC ALLISTER  MIKE
LONGLASTNAMETHREE  JIM
;
```

The colon is unnecessary in this case because the ampersand implies the colon's function. The value for LASTNAME is read from the first nonblank column up to 15 columns or until two consecutive blanks or the end of the line.

Operating systems: All

Note: using the : or & format modifier invokes the scanning feature of list input. Although the variable **value** contains only the input read from the next nonblank column until either the next blank or the defined length of the variable, the input pointer does not stop until it finds a blank. Therefore, do not use the : or the & format modifier unless

- input data values are separated by blanks (two blanks in the case of the & modifier), or
- you intend to read only the last variable in the input line using the format modifier, and therefore do not care about the input pointer location.

Order of variables Simple formatted input (that is, without pointer controls) requires that the variables be in the same order as their corresponding values in the input data. You can read variables in any order by using pointer controls.

Missing values Missing values in formatted input are generally represented by blanks or a single period for a numeric value, and by blanks for a character value. However, some informats, such as $CHAR., treat blanks as valid data. See the chapter "SAS Informats and Formats" for more information on how a particular informat handles data.

Blanks The informat used with formatted input determines the way blanks are interpreted. For example, the $CHAR. informat reads blanks as part of the value, whereas the BZ. informat turns blanks into zeros.

Storing informats Informat names specified in the INPUT statement are not stored with the SAS data set. Informat names specified with the INFORMAT or ATTRIB statement are stored, allowing you to input informatted values in a later DATA step without specifying the informat or to input data using PROC FSEDIT.

Advanced Input Statement Features

As the SAS System reads values from the input lines, it keeps track of its position with a pointer. For example, if each observation has several data lines of input and you are reading a value that begins in the tenth column of the second line, the pointer's position is line 2, column 10.

Pointer controls reset the pointer's column and line position, and tell the INPUT statement where to go to read the data value. There are two kinds of pointer controls, and two line hold specifiers:

1. **column pointer controls** move the pointer to the column you specify. There are four column pointer controls: *@n*, *@pointvariable* , *+n*, and *+pointvariable*.
2. **line pointer controls** move the pointer to the line number specified. There are five line pointer controls: *#n*, *#pointvariable*, */*.
3. **line hold specifiers** keep the line pointer on the current input line. There are two line hold specifiers: *trailing* @ and *trailing* @@.

You can also determine the pointer's current column and line location with the the COLUMN= and LINE= options in the INFILE statement.

Specify pointer controls before the variable to which they apply. Line pointer controls at the end of the INPUT statement can be used to move to the next input line or define the number of input lines per observation. The sections below describe each type of column and line pointer control and line hold specifier.

Column pointer controls Column pointer controls indicate in which column an input value starts. There are four column pointer controls. The first two begin with an @ sign and are absolute; that is, they move the pointer to a specified column number. The last two begin with a + sign and are relative; that is, they move the pointer a specified number of columns.

@n
: moves the pointer to column *n*.[4]

@pointvariable
: moves the pointer to the column given by the value of *pointvariable*.

  To move the pointer to a specific column, use the @ followed by the column number or by a variable whose value is that column number. For example, the statement

```
INPUT @15 SALES;
```

  moves the pointer to column 15. The INPUT statement in this example

```
DATA ONE;
   A=25;
   INPUT @A NAME $10.;
more SAS statements
```

also moves the pointer to column 25, the value of A. The @pointvariable control is often combined with the trailing @ control described in **Line Pointer Controls**. In this example,

```
DATA ONE;
   INPUT X @;
   IF 1 <= X <= 10 THEN
      INPUT @X CITY $12.;
   ELSE
      INPUT @50 COUNTY $10.;
 more SAS statements
   CARDS;
```

the SAS System obtains the value of X for the current observation in the first INPUT statement and uses that value to determine the column to move to in the second INPUT statement. See the section **Line Pointer Controls**.

The pointer can go backward as well as forward. For example, this INPUT statement

```
INPUT @26 BOOK @1 COMPANY;
```

first reads a value for BOOK starting at column 26 and then moves back to column 1 on the same line to read a value for COMPANY. If a negative number is associated with the @ pointer control, the pointer moves to column 1.

In this example that mixes input styles,

```
INPUT NAME $ 1-10 @15 PULSE 3. @20 WAIST 2. AGE;
```

a value for NAME is read with column input from columns 1 through 10; the pointer moves to column 15, reads a value for PULSE from 15, 16, and 17 and for WAIST from 20 and 21, both with formatted input, and then reads a value for AGE with list input.

+*n*

moves the pointer *n* columns.[5]

+*pointvariable*

moves the pointer the number of columns given by the *pointvariable* value.

The relative pointer control (+), followed by a number or a variable, moves the pointer by the number or the variable's value. To move backward, use the +pointvariable. For example, the statement

```
INPUT @23 LENGTH 4. +5 WIDTH;
```

moves the pointer to column 23, reads a value for LENGTH from the next four columns (23, 24, 25, and 26), and then advances the pointer five columns to read a WIDTH value in column 32.

You can move the pointer backwards by setting a *pointvariable* in the DATA step to the number of columns you want to back up and by specifying that variable after the + pointer control. For example, say you want to back up one column:

```
DATA FOUR;
   M=-1;
   INPUT X 1-10 +M Y;
```

This INPUT statement reads a value for X from columns 1 through 10 and then moves the pointer back one column to read a value for Y starting in column 10.

Column pointer location after reading The pointer's location after reading depends on the input style used. When reading with **list input**, the pointer moves to the second column after the value. For example, suppose the statement

```
INPUT X Y;
```

reads a value for X from columns 1 and 2. Because the value for X was read with list input, the pointer's location is column 4, the second column after the value. The SAS System reads the Y value starting from column 4.

Exception: When the ampersand (&) modifier is used with list input, the pointer moves to the third column after the value since the ampersand requires two spaces between input values.

When reading with either **column** or **formatted input**, the pointer moves to the first column after the value. For example, consider the statement

```
INPUT A 3-4 B;
```

or the statement

```
INPUT @3 A 2. B;
```

The field for the A value ends in column 4 (even if the value itself occupies only column 3), and the pointer moves to column 5 after reading the value for A.

Whenever you move the pointer to a new input line, the column pointer is automatically set to 1.

Line pointer controls Line pointer controls specify the input line from which the INPUT statement is to read a value.

The following three line pointer controls are used when each observation has values on more than one input line:

#n

moves the pointer to line *n*. [6]

#pointvariable

moves the pointer to the line given by the value of *pointvariable*.

When the INPUT statement needs to read several input lines per observation, the SAS System must know from which line to read values. The number following the # pointer control specifies the line that contains the next group of values. For example, in the statement

```
INPUT @12 NAME $10. #2 ID 3-4;
```

a #1 is implied between INPUT and the @12 pointer direction, indicating that the value for NAME begins in column 12 of the first input line. The value for ID is in columns 3 and 4 of the second input line. The highest number following the # pointer control in the INPUT statement determines how many lines per observation are read unless you override this by specifying N= in the INFILE statement. For example, in this statement,

```
INPUT @31 AGE 3. #3 ID 3-4 #2 @6 NAME $20.;
```

the highest value after the # is 3; thus, the INPUT statement reads three input lines each time it executes, unless the N= option has been specified in the associated INFILE statement.

When each observation has multiple input lines, but values are not read from the last line, a # pointer control at the end of the INPUT statement must move the pointer to the last line in each observation unless the N= option has been specified in the INFILE statement. For example, if there are four lines per observation but values are read only from the first two lines, the INPUT statement may look like this:

```
INPUT NAME $ 1-10 #2 AGE 13-14 #4;
```

/

advances the pointer to column 1 of the next input line. For example, the statement

```
INPUT AGE GRADE / SCORE1-SCORE5;
```

reads values for AGE and GRADE from one input line, and then skips to the next line to read values for SCORE1-SCORE5.

When you have advanced to the next line with the / pointer control, you must use the #n pointer control or the INFILE statement N= option to define the number of lines per observation in order to move the pointer back to an earlier line. To return to an earlier line when reading from multiline input, give the total number of lines in the input record with the #n pointer control at the end of the INPUT statement or the N= option in the INFILE statement. The following statement requires the #2 pointer control unless the INFILE statement has the N= option specified:

```
INPUT A / B #1 @52 C #2;
```

The above statement reads a value for A from the first line, for B from the second, and then returns to the first line to read a value for C. The #2 pointer control identifies two input lines for each observation, so the pointer can return to the first line for the value of C.

If the number of input lines per observation varies, use the N= option in the INFILE statement to give the maximum number of lines per observation. See the INFILE statement description for more information. The N= option overrides any specification implied by pointer controls.

Line hold specifiers Line hold specifiers keep the line pointer on the current input line when:

- a data line is read by more than one INPUT statement (*trailing* @).
- one input line has values for more than one observation (*trailing* @@).

Use the following line hold specifier to read an input line with more than one INPUT statement:

@, trailing

holds a data line for the next INPUT statement in the step. The next INPUT statement for the same execution of the DATA step accesses the same data line rather than reading a new one.

The @ is called a *trailing at-sign* because it must be the last item in the INPUT statement.

Normally, each INPUT statement in a DATA step reads a new data line. To read values from the same data line with more than one INPUT statement, use a trailing @ at the end of each INPUT statement that is to read from that line.

For example, say that you have two kinds of input data lines. One type of data line gives information about a particular college course; the other contains information about the students taking that course. You need two INPUT statements to read the two lines because they have different variables and different formats. Lines containing class information have a C in column 1; lines containing student information have an S in column 1. You need to check each line as it is read to know which INPUT statement to use. You need an INPUT statement that reads only the variable telling whether the line is a student or class record:

```
DATA SCHEDULE;
   INPUT TYPE $ 1 @;
   IF TYPE='C' THEN INPUT COURSE $ PROF $;
   ELSE IF TYPE='S' THEN INPUT NAME $ ID;
```

The first INPUT statement reads the TYPE value from column 1 of every line. Since this INPUT statement ends with a trailing @, the next INPUT statement in the DATA step reads the same line. The IF statements that follow check whether the line is a class or student line, and each gives an INPUT statement to read the rest of the line.

A line held with a trailing @ can be released with an INPUT statement without operands or a trailing @:

```
INPUT;
```

This statement releases the current data line so that the next INPUT statement reads a new line.

An input line held by a trailing @ is automatically released when the DATA step executes again. Thus, a RETURN or DELETE statement releases the line. If you want to read the same input line in more than one execution of the DATA step, use the @@ symbol, described below.

Use the following line hold specifier when a single input line contains values for more than one observation:

@@, trailing
holds an input line for further executions of the DATA step. The @@ symbol (called a *double trailing at-sign*) is useful when each input line contains values for several observations.

For example, say that you have input with each line containing several NAME and AGE values. You want to read first a NAME value, then an AGE value, then output the observation; then read another NAME and another AGE value to output, and so on until you have read and output all the input values in the line. Use a double trailing @ in your INPUT statement:

```
DATA THREE;
   INPUT NAME $ AGE @@;
   CARDS;
JOHN 13 MARY 12 SUE 15 TOM 10
;
```

The SAS System releases a line held by a trailing @@ when the pointer moves past the end of the line. In addition, an INPUT statement without operands or a trailing @ releases the held line immediately

```
INPUT;
```

and an INPUT statement with a single trailing @ releases the held line at the end of the current iteration of the DATA step (just as if @@ had not been specified):

```
INPUT @;
```

Reading past the end of a line When @ or + pointer controls are used with a value that moves the pointer to or past the end of the current line and the next value is to be read from the current column, SAS goes to column 1 of the next line to read it. It also prints the message on the SAS log:

```
NOTE: SAS WENT TO A NEW LINE WHEN INPUT STATEMENT REACHED
      PAST THE END OF A LINE.
```

Use the STOPOVER option in the INFILE statement if you want to treat this condition as an error and stop building the data set.

You can also use the MISSOVER option in the INFILE statement to set the remaining INPUT statement variables to missing values if the pointer reaches the end of a line. See the INFILE statement description in this chapter for more information.

Formatted lists with pointer controls You can write shorter INPUT statements by using pointer controls in format lists. Format lists can contain any of the pointer controls (@, @@, #, /, and +). For example, suppose you want to read 20 LOC values and 20 AMOUNT values for each observation. The values are arranged in the input lines as a string of pairs, with the LOC value followed by an AMOUNT value. Rather than write a long INPUT statement listing all 40 variables, you can use a format list to match the LOC values in columns 1-2, 4-5, 7-8, and so on with the AMOUNT values in columns 3, 6, 9, and so on. The following INPUT statement reads the values:

```
INPUT (LOC1-LOC20) (2. +1) @1 (AMOUNT1-AMOUNT20) (+2 1.);
```

After the LOC values have been read, the @ pointer control moves the pointer to column 1 in the input line to read the AMOUNT values.

SAS format modifiers for error reporting SAS format modifiers for error reporting define the amount of information printed in the log when the SAS System encounters an error in an input value.

?

suppresses the invalid data message that the SAS System prints when it encounters an invalid data value. For example:

```
INPUT X ? 10-12;
INPUT (X1-X10) (? 3.1);
```

When SAS encounters an invalid character in a value for the variable X, the SAS System takes the actions described in the later section **How the SAS System Handles Invalid Data**, except it does **not** print the invalid data message.[7]

??

the double question mark suppresses the printing of both the error messages and the input lines when invalid data values are read. The ? and ?? message modifiers both suppress the invalid data message. The ?? modifier also prevents _ERROR_ from being set to 1 when invalid data are read. Thus, the statement

```
INPUT X ?? 10-12;
```

is equivalent to

```
INPUT X ? 10-12;
_ERROR_=0;
```

Invalid X values are still set to missing values.

Named Input

The named input style can be used when your data lines contain variables followed by an equal sign and a value for the variable.[3] For example, you would use named input to read an input line containing AGE=21, rather than just the value 21, for the numeric variable AGE.

Restrictions When an equal sign (=) follows an INPUT statement variable, the SAS System expects the remaining input line data to contain **only** named input. This means that

- the remaining input values must be in the form *variable=value*.
  If any of the values are not in named input form, the SAS System takes the action described in the later section, **How the SAS System Handles Invalid Data**.
  If named input values continue after the end of the current data line, a / at the end of the data line tells the SAS System to go to the next line and continue reading with named input. Note that a / in the INPUT statement has the same effect as a / in the input data line with named input.
- You cannot switch to another input style for a particular data line once you start reading it with named input. All of the remaining values on the data line must be in the named input form.
  However, you can read input data in other forms with corresponding input styles **before** starting to use named input.
- If you have used a variable that appears in a line of named input in any other statement (for example, in a LENGTH, ATTRIB, FORMAT, or INFORMAT statement), the variable is automatically included in the data set, regardless of whether it is explicitly specified in an INPUT statement. Even if you explicitly read only one named input value with the INPUT statement, all of the remaining named input values in the current record are read, and the corresponding variables are created.
- Use an INFORMAT statement preceding the INPUT statement to read a named input variable with an informat. If you supply any INFORMAT statements, you must supply one for each informat used in the lines to be read with named input.
- You cannot use the *arrayname(*)* variable specification with named input.

The form of the INPUT statement for reading variables with named input is distinguished by the equal sign following a variable name:

INPUT *[pointercontrol] variable*=[**$**] *[informat]*;

where

=

indicates that the named input style is to be used.
The INPUT statement begins reading named input at the current location of the input pointer. Thus, if the input lines include some

data values at the beginning of the line that cannot be read with named input, you can use another input style to read them. For example, the INPUT statement

```
INPUT PULSE WAIST AGE= SEX=;
```

reads the data line

```
80 32 AGE=35 SEX=M
```

Once an INPUT statement starts reading named input (initiated by an equal sign after a variable), SAS expects all remaining values in the input line to be in this form. The named input values can be in any order and all do not need to be specified in the INPUT statement. For example,

```
INPUT ID NAME=$20. SEX=$ AGE=;
```

could be used to read the input line

```
4798 AGE=23 SEX=F NAME=JOHN SMITH
```

In this case, the variable ID is read with ordinary list input. Then the remaining values in the input line are read with named input.

If character values in the input lines contain an equal sign, put two blanks before and after the data value. For example, the input line

```
HEADER=  AGE=60 AND UP  NAME=JOHN DOE
```

could be read with the statements

```
FORMAT HEADER $30. NAME $15.;
INPUT HEADER= NAME=;
```

because there are two blanks before and after the value for the variable HEADER, which contains an equal sign.

Special Topics

The INPUT statement without operands An INPUT statement without variable names is a statement without operands. This type of INPUT statement has several uses:

- to bring an input data line into the DATA step without creating any SAS variables
- to copy an input record to the output file
- to release an input line held by a trailing @ or @@.

For example, the following lines copy the input file to the output file without creating any SAS variables:

```
DATA _NULL_;
   INFILE IN;
   FILE OUT;
   INPUT;
   PUT _INFILE_;
```

How the SAS System handles invalid data An input value is invalid if it

1. requires an informat that is not available at execution time
2. does not conform to the informat specified
3. cannot be read with the input style specified
4. is read as standard numeric data (no $ sign or informat) and does not conform to the rules for standard SAS numbers.

When an error in an input value is encountered, the SAS System

- sets the value of the variable being read to missing.
- prints the input line and column number containing the invalid value on the SAS log.
- sets the automatic variable _ERROR_ to 1 for the current observation.
- prints an invalid data message.
- prints the input lines corresponding to the current observation. If a line contains unprintable characters, it is printed in hexadecimal form. A scale is printed above the input line to help determine column numbers.

See the "SAS Log and Procedure Output" chapter for an example.

End-of-file End-of-file occurs when an INPUT statement reaches the end of the data. When a DATA step tries to read another record after end-of-file has been reached, the DATA step execution stops. You can detect end-of-file using the INFILE options END= and EOF=, and stop executing INPUT statements for that INFILE if you want to continue executing the DATA step. (See the **INFILE** statement for more details.) However, you cannot use END= or EOF= on files read from the terminal (including CARDS), or for files for which you specified the UNBUFFERED option.

Arrays An array member, either explicitly or implicitly subscripted, can be read with an INPUT statement.[8] However, an array name cannot be used for such things as a column pointer.

When reading an explicitly subscripted array, specify the subscript following the array name:[9]

```
DATA TWO;
     .
     .
     .
INFILE ARRAY2
INPUT ARR1 [INDX+2] ... ;
```

In this case the subscript can be any valid SAS expression. The expression must evaluate to a valid subscript value at the time the INPUT statement executes.[9]

Notes

1. **CMS, OS, VM/PC, and VSE**: The period preceding the decimal specification is optional.

2. **AOS/VS, PRIMOS, and VMS**: The *n** modifier is not available.

3. Named input can be used only under **CMS, OS, VM/PC,** and **VSE.**

4. **AOS/VS, PRIMOS, and VMS**: *n* can be any expression in parentheses that evaluates to a positive integer.

5. **AOS/VS, PRIMOS, and VMS**: *n* can be any expression in parentheses, including a negative number.

   CMS, OS, VM/PC, and VSE: The number following the + must be a positive integer.

6. See note 4.

7. **CMS, OS, VM/PC, and VSE**: To suppress printing the input lines containing invalid characters, reset the value of the _ERROR_ automatic variable back to 0 with this statement

```
_ERROR_=0;
```

after your INPUT statement. Invalid X values are still set to missing values.

8. Implicitly subscripted arrays can be read only under CMS, OS, VM/PC, and VSE.

When reading an implicitly subscripted array, the index variable defined for the array (or the automatic index, _I_) determines which element of the array to input. The subscript must contain a valid value at the time the INPUT statement executes. For example, the following statements read and write values for the five elements of the array OLD:

```
DATA ONE;
   ARRAY OLD(I) $ E1-E5;
   DO OVER OLD;
      INPUT OLD @;
      PUT OLD;
      END;
   CARDS;
a b c d e
e f g h i
;
```

9. **AOS/VS, PRIMOS, and VMS**: Specify the subscript in braces {}, or brackets [ ].

CMS, OS, VM/PC, and VSE: Specify the subscript in parentheses () or brackets [ ].

10. **CMS, OS, VM/PC, and VSE**: An asterisk specified for the subscript tells the SAS System to read the entire array:

```
INPUT ARRAY2(*) ... ;
```

The effect is the same as if you had listed all of the individual array members in a variable list.

KEEP Statement

Operating systems: All

You can use the KEEP statement in a DATA step to specify the variables that are to be included in any SAS data sets being created. The KEEP statement applies to all data sets being created in the step. To keep variables in particular data sets when more than one data set is being created in the DATA step, use the KEEP= data set option with each data set name. (See "SAS Files" for more about data set options.)

If a DATA step includes a KEEP statement, only variables appearing in the KEEP statement are included in new data sets. Variables not listed in the KEEP statement remain available for use in program statements. The KEEP statement can appear anywhere among the program statements in the DATA step; it is not an executable statement.

The form of the KEEP statement is

KEEP *variables*;

where

variables specifies the variables you want included in the data set or data sets being created. Any form of variable list may be used (see "Introduction to the SAS Language" for details). Do not abbreviate the variable names.

Here is an example:

```
DATA AVERAGE;
   INPUT NAME $ SCORE1-SCORE20;
   AVG=MEAN(OF SCORE1-SCORE20);
   KEEP NAME AVG;
   CARDS;
data lines
;
```

The effect of the KEEP statement is the reverse of the DROP statement's effect. To save writing, the KEEP statement is preferred if fewer variables are being kept than dropped.

Do not use both KEEP and DROP statements in the same step. When both RENAME and KEEP statements are used in a DATA step, the KEEP statement is applied first. This means that the old name should be used in the KEEP statement.

In Version 5 the KEEP statement is available only in the DATA step. To include variables in a PROC step, use the KEEP= data set option.

LABEL Statement

Operating systems: All

You can use LABEL statements in a DATA step to give labels to variables. The label is stored with the variable name in the SAS data set and printed by many SAS procedures.

The form of the LABEL statement is:

LABEL *variable='label'...;*

where

variable names the variable to be labeled.

label specifies a label of up to 40 characters including blanks. The label must be enclosed in either single or double quotes. (If double quotes are used, the SAS system option DQUOTE must be in effect.) Single quotes as part of the label must be written as two single quotes and are counted as one character.

Any number of variable names and labels can appear. Here are examples of LABEL statements:

```
LABEL COMPOUND='TYPE OF DRUG';
LABEL SCORE1="GRADE ON APRIL 1 TEST"
      SCORE2="GRADE ON MAY 1 TEST";
LABEL DATE='IF Y=0 W=DATE OF TEST';
LABEL N='MARK''S EXPERIMENT NUMBER';
LABEL N="MARK'S EXPERIMENT NUMBER";
```

Labels, Statement

Operating systems: All

You can use a statement label to identify a statement referred to by a GO TO or LINK statement. A statement label has the form

label: *statement*;

These terms make up the statement label:

label identifies the destination of a GO TO statement, a LINK statement, the HEADER= option in a FILE statement, or the EOF= option in an INFILE statement. The label can be any valid SAS name followed by a colon (:).

statement is any executable statement in the same DATA step as the statement or option that references it. No two statements in a DATA step should have the same label. If a statement in a DATA step is labeled, it should be referenced by a statement or option in the step.

For example:

```
DATA INVENTRY ORDER;
   INPUT ITEM $ STOCK @;
   IF STOCK=0 THEN GO TO REORDR;
   OUTPUT INVENTRY;
   RETURN;
REORDR: INPUT SUPPLIER $;
   PUT 'ORDER ITEM # ' ITEM 'FROM ' SUPPLIER;
   OUTPUT ORDER;
   CARDS;
data lines
;
```

In the example above, the first INPUT statement reads a record containing an item description (ITEM) and the number in stock (STOCK). If STOCK=0, the GO TO statement causes the SAS System to jump to the statement labeled REORDR—another INPUT statement. SAS reads the name of the supplier for that item, writes a message on the log, and outputs the record to data set ORDER. When STOCK is not zero, the record is output to data set INVENTRY, and SAS returns to the beginning of the DATA step for a new observation.

LENGTH Statement

Operating systems: All

Introduction

You can include a LENGTH statement in a DATA step to specify the number of bytes the SAS System is to use for storing values of variables in each data set being created.

Specify the LENGTH statement as

LENGTH [*variables* [**$**] *length*] ... [**DEFAULT=***n*];

These terms are included in the LENGTH statement:

variables names the variable or variables to which you want to assign a length. The variable list can include any variables in the data set; an array reference may not appear.

$ indicates that the variable or variables in the preceding list are character variables.

length is a numeric constant that can range from 2 to 8 for numeric variables[1] and from 1 to 200 for character variables. Note that this length value is not a format; it does not contain a period.

DEFAULT=*n* optionally, changes the default number of bytes used for storing the values of newly created numeric variables from 8 to the number *n* that you specify. *N* can range from 2 to 8.[2]

For example, the statement

```
LENGTH NAME $ 20;
```

sets the length of the character variable NAME to 20.

The length of a variable depends on

- whether the variable is numeric or character
- how the variable was created
- whether a LENGTH statement is present.

The following discussion describes variable lengths in general. The **Assignment Statement** earlier in this chapter contains a table that summarizes the length of variables created in assignment statements. The "SAS Functions" chapter discusses lengths of values returned by functions.

Numeric Data

Normally, numeric variables in SAS data sets have a length of 8 bytes. However, many values can be represented exactly in fewer than 8 bytes. When your data set is very large, using fewer than 8 bytes to store values that do not need that much precision can significantly decrease external storage requirements. Before you use the LENGTH statement to change the number of bytes for storing numeric values, however, note carefully the discussion below on truncation problems.[3]

Truncation Consider the case where numeric values are represented in a base-16 number system. Exact decimal fractions, such as .3, are not necessarily exact fractions in base-16 representations. This situation can create difficulties.

For example, suppose that you use a LENGTH statement to store the values of a variable in 4 bytes:

```
LENGTH A 4;
```

Each value of A is initially moved into an 8-byte field during the DATA step. When the value is moved to the SAS data set, the last 4 bytes of the value are dropped. Then, when the value is used for processing in a later DATA or PROC step, its representation is again brought up to 8 bytes by appending nonsignificant zeros.

Unless the part of the representation originally dropped consists of all zeros, something is lost in truncation. The example below illustrates how that loss can affect the behavior of the SAS System:

```
DATA ONE;
   INPUT A 1-4 B 6;
   LENGTH DEFAULT=4;
   CARDS;
1.4  6
1.1  5
1.1  6
1.3  4
1.3  3
2.0  4
;
DATA TWO;
   SET ONE;
   IF A=1.3;
RUN;
```

Data set TWO has **no** observations. The constant 1.3 in the subsetting IF statement in the second DATA step has the full 8-byte representation of the 1.3, while the fourth and fifth values of A are identical to it only in the first 4 bytes of their values. Hence A will never be found equal to 1.3.

Although you should be aware of problems like these, using single-precision storage of 4 bytes does give you 7 significant digits, which is sufficient for most applications. Thus, using a default length of 4 to store numeric values when space is an important consideration is usually safe. See "SAS Expressions" for additional information on working with calculated fractional values.

Character Data

The length of a character variable is set the first time that the variable is used in a SAS DATA step. After the length has been specified, you cannot change it

except in a later DATA step using a LENGTH statement. (See **Changing Variable Lengths** below.)

Since the INPUT statement can implicitly define a character variable's length, the LENGTH statement should precede the INPUT statement when it defines lengths for character variables that are different from the lengths implied in the INPUT statement.

For example, the statements

```
DATA ONE;
   INPUT NAME $ 1-10;
   CARDS;
data lines
;
```

implicitly assign the variable NAME a length of 10. If a LENGTH statement appears before the INPUT statement, as in

```
DATA TWO;
   LENGTH NAME $20;
   INPUT NAME $ 1-10;
   CARDS;
data lines
;
```

NAME's length in the output data set is 20 instead of 10.

When a character variable appears for the first time in a DATA step, its length is determined from the context of its use. For example, consider these statements:

```
DATA TWO;
   INPUT X;
   IF X=1 THEN A='NO';
   ELSE A='YES';
```

A appears for the first time in the assignment A='NO'. Thus A's length is 2 in the data set, the length of the character literal 'NO'. When the value 'YES' is assigned to A, only the first 2 letters are saved; the 'S' is lost. To avoid this problem, use a LENGTH statement to give A the length you want:

```
DATA TWO;
   INPUT X;
   LENGTH A $ 3;
   IF X=1 THEN A='NO';
   ELSE A='YES';
```

or rearrange the statements:

```
DATA TWO;
   INPUT X;
   IF X¬=1 THEN A='YES';
   ELSE A='NO';
```

A's length in both data sets is 3, and so the complete value 'YES' can be saved.

In another example, below:

```
DATA THREE;
   LENGTH B $ 15;
   INPUT X B;
   IF X=1 THEN A=B;
```

B's length is defined as 15 by the LENGTH statement. The first appearance of A in the step is in the assignment A=B. Thus A's length is 15, determined by the length of B.

When you use list input to read a character variable, a length of 8 is assumed. If any of the values are longer than 8, they are truncated to 8 unless a LENGTH statement defines a longer length, as in the previous example.

Also note that when you use a LENGTH statement for a character variable before the INPUT statement, you need not specify the dollar sign ($) in the INPUT statement since SAS knows that the variable is character by the time it encounters the INPUT statement.

Changing Variable Lengths

The contents of SAS data set FIRST show the length and type of its two variables B and X: B is character of length 10; X is numeric of length 8.

Output 4.2 Length of a Numeric and a Character Variable

```
                                    CONTENTS PROCEDURE


                                CONTENTS OF SAS DATA SET WORK.FIRST

CREATED BY OS JOB LENGTH    ON CPUID XX-XXXX-XXXXXX   AT 20:23 TUESDAY, JANUARY 8, 1985   BY SAS RELEASE 5.XX
DSNAME=XXXXXXXX.XXXXXXX.XXXXX.XXXXXXX.XXXXXXXX   BLKSIZE=19056   LRECL=22   OBSERVATIONS PER TRACK =866
GENERATED BY DATA
NUMBER OF OBSERVATIONS: 3   NUMBER OF VARIABLES: 2

                              ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  B         CHAR     10        4
  2  X         NUM       8       14


                    ------------------------------ SOURCE RECORDS ------------------------------
                    | DATA FIRST;                                                               |
                    |    LENGTH B $ 10;                                                         |
                    |    INPUT X B;                                                             |
                    |    CARDS;                                                                 |
                    -----------------------------------------------------------------------------
```

To change the length of B, a character variable, you must create a new data set and precede the SET statement with a LENGTH statement. You can also change X's length in the LENGTH statement:

```
DATA SECOND;
   LENGTH B $ 8 X 4;
   SET FIRST;
```

The variables have different lengths in data set SECOND as shown in **Output 4.3**.

Output 4.3 Different Length for Variables in New Data Set

```
                                        CONTENTS PROCEDURE


                                  CONTENTS OF SAS DATA SET WORK.SECOND

CREATED BY OS JOB LENGTH    ON CPUID XX-XXXX-XXXXXX   AT 20:23 TUESDAY, JANUARY 8, 1985   BY SAS RELEASE 5.XX
DSNAME=XXXXXXXX.XXXXXXX.XXXXX.XXXXXXX.XXXXXXXX   BLKSIZE=19060   LRECL=16   OBSERVATIONS PER TRACK =1191
GENERATED BY DATA
NUMBER OF OBSERVATIONS: 3   NUMBER OF VARIABLES: 2

                              ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  B         CHAR      8        4
  2  X         NUM       4       12


                    ------------------------------ SOURCE RECORDS ------------------------------
                    | DATA SECOND;                                                              |
                    |    LENGTH B $ 8 X 4;                                                      |
                    |    SET FIRST;                                                             |
                    ------------------------------------------------------------------------------
```

Although a character variable's length must be changed by placing a LENGTH statement **before** the SET, you can change X's length in a LENGTH statement placed anywhere in the step.

```
DATA THIRD;
   SET FIRST;
   LENGTH X 4;
```

The result is shown in **Output 4.4**.

Output 4.4 Different Length for Numeric Variable

```
                                        CONTENTS PROCEDURE


                                  CONTENTS OF SAS DATA SET WORK.THIRD

CREATED BY OS JOB LENGTH    ON CPUID XX-XXXX-XXXXXX   AT 20:23 TUESDAY, JANUARY 8, 1985   BY SAS RELEASE 5.XX
DSNAME=XXXXXXXX.XXXXXXX.XXXXX.XXXXXXX.XXXXXXXX   BLKSIZE=19066   LRECL=18   OBSERVATIONS PER TRACK =1059
GENERATED BY DATA
NUMBER OF OBSERVATIONS: 3   NUMBER OF VARIABLES: 2

                              ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  B         CHAR     10        4
  2  X         NUM       4       14


                    ------------------------------ SOURCE RECORDS ------------------------------
                    | DATA THIRD;                                                               |
                    |    SET FIRST;                                                             |
                    |    LENGTH X 4;                                                            |
                    ------------------------------------------------------------------------------
```

The same rules apply to changing lengths of variables in any previously existing SAS data set, whether read with a SET, MERGE, or UPDATE statement.

Notes

1. **PRIMOS:** The minimum length of a numeric variable is 3.
2. **PRIMOS:** *N* can range from 3 to 8.
3. **AOS/VS, PRIMOS, and VMS**: See the SAS Companion for your operating system for additional information on variable lengths.

CMS, OS, VM/PC, and VSE: The discussion of truncation problems applies to data values that are not whole numbers (integers). When a variable's values are all integers, you can safely use the LENGTH statement to save storage space for the data set being created. **Table 4.6** shows, for each possible length, the minimum number of significant digits that can be represented in that length and the magnitude of the largest integer that can be represented exactly in that length.

Table 4.6 Significant Digits and Largest Integer by Length: CMS, OS, VM/PC, and VSE

Length	Significant Digits Retained	Largest Integer Represented Exactly
2	2	255
3	4	65,535
4	7	16,777,215
5	9	4,294,967,295
6	12	1,099,511,627,775
7	14	281,474,946,710,655
8	16	72,057,594,037,927,935

LINK Statement

Operating systems: All

A LINK statement tells the SAS System to jump immediately to the statement label indicated in the LINK statement and to continue executing statements from that point until a RETURN statement is executed. The RETURN statement causes SAS execution to return to the statement immediately following the LINK statement and continue from there. The LINK statement and the destination must be in the same DATA step. The destination is identified by a statement label in the LINK statement and the destination statement.

The statement has the form:

LINK *label*;

where

label specifies a statement label that identifies the LINK destination. See **Labels, Statement** earlier in this chapter for more information.

The difference between the LINK and GO TO statements is in the action of a subsequent RETURN statement. A RETURN after a LINK returns SAS execution to the statement following the LINK; a RETURN statement after a GO TO returns SAS execution to the beginning of the DATA step. In addition, a LINK statement is usually used with an explicit RETURN statement; a GO TO statement is often used without a RETURN statement. In that case execution continues until another GO TO statement or the end of the DATA step (which contains an implied RETURN) is reached.

You can place another LINK statement within a LINKed routine (called *nesting*). The maximum level of nesting is 10 (that is, 10 LINK statements with no intervening RETURN statements). When more than one LINK statement has been executed, a RETURN statement tells the SAS System to return to the statement following the last LINK statement executed.

Here is an example using one LINK statement:

```
*STUDY OF WATER RESOURCES;
DATA HYDRO;
   INPUT TYPE $ WD STATION $;
   LABEL TYPE='STATION TYPE'
      WD='DEPTH TO WATER';
   ELEV=.;
   IF TYPE ='ALUV' THEN LINK CALCU;
   YEAR=1985;
   RETURN;
CALCU: IF STATION='SITE_1' THEN ELEV=6650-WD;
   IF STATION='SITE_2' THEN ELEV=5500-WD;
   RETURN;
   CARDS;
ALUV 523 SITE_1
UPPA 234 SITE_2
ALUV 666 SITE_2
more data lines
;
```

When the value of TYPE is ALUV, SAS executes the LINK statement, jumps to label CALCU, and executes that statement and the next two. The RETURN statement after the second IF statement causes SAS to return to the next statement, in this case

```
YEAR=1985;
```

When SAS reaches the RETURN statement after the assignment statement, SAS outputs that observation and returns to the top of the DATA step for a new data line.

If you need to execute a group of statements at only one point in a program, a DO group is simpler than a LINK-RETURN group. For example, you can write the DATA step above as

```
DATA HYDRO;
   INPUT TYPE $ WD STATION $;
   LABEL TYPE='STATION TYPE'
      WD='DEPTH TO WATER';
   ELEV=.;
   IF TYPE ='ALUV' THEN DO;
      IF STATION='SITE_1' THEN ELEV=6650-WD;
      IF STATION='SITE_2' THEN ELEV=5500-WD;
      END;
   YEAR=1985;
   CARDS;
data lines
;
```

See the DO statement description for more information.

If you need to execute a group of statements at several points in a program, a LINK statement can simplify coding the program and make it easier to understand. The example below links to a statement that recodes grades of 'E' to 'F' for grades on three tests:

```
DATA CLASS;
   INPUT ID TEST1 $ TEST2 $ TEST3 $;
   TEST=TEST1;
   LINK RECODE;
   TEST1=TEST;
   TEST=TEST2;
   LINK RECODE;
   TEST2=TEST;
   TEST=TEST3;
   LINK RECODE;
   TEST3=TEST;
   RETURN;
RECODE: IF TEST='E' THEN TEST='F';
   RETURN;
   CARDS;
data lines
;
```

To recode each test grade, SAS moves the grade to a variable TEST, links to RECODE and recodes it, and then moves the recoded value to the original variable.

You can also write this step with an ARRAY statement:

```
DATA CLASS;
```

```
   ARRAY TEST[3] $ TEST1-TEST3;
   INPUT ID TEST1 $ TEST2 $ TEST3 $;
   DO I=1 TO DIM(TEST);
      IF TEST[I]='E' THEN TEST[I]='F';
      END;
   CARDS;
data lines
;
```

See the ARRAY statement description for more information.

LIST Statement

Operating systems: All

You can use the LIST statement to list on the SAS log the input data lines for the observation being processed. When the LIST statement is executed, the SAS System sets a flag that causes the current input lines to be printed at the end of the DATA step. A ruler indicating columns appears before the first line listed.

The form of the LIST statement is

LIST;

The LIST statement is useful for printing suspicious input lines read by an INPUT statement. Here is an example:

```
DATA EMPLOYEE;
   INPUT SSN 1-9 #3 W2AMT 1-6;
   IF W2AMT=. THEN LIST;
   CARDS;
123456789
JAMES SMITH
356.79
345671234
JEFFREY THOMAS
.
;
```

Each time W2AMT is missing, SAS prints the three current input data lines on the SAS log. **Output 4.5** shows the log for the DATA step above.

Output 4.5 Listing Input Data Lines

```
1          SAS(R) LOG   OS SAS 5.XX           MVS/XA JOB LIST1    STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB LIST1 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

1           DATA EMPLOYEE;
2              INPUT SSN 1-9 #3 W2AMT 1-6;
3              IF W2AMT=. THEN LIST;
4              CARDS;


RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

8           345671234
9           JEFFREY THOMAS
10          .
NOTE: DATA SET WORK.EMPLOYEE HAS 2 OBSERVATIONS AND 2 VARIABLES. 953 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.95 SECONDS AND 372K.

11          ;
NOTE: SAS USED 376K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The DATA step below illustrates how SAS prints lines from a LIST statement at the end of the DATA step. The step includes both a LIST statement and a PUT statement. Lines printed by the PUT statement are printed first on the log.

```
DATA D;
   INPUT X Y Z;
   LIST;
   IF X=Y THEN PUT 'X EQUALS Y ' X=;
   CARDS;
1 2 3
2 2 4
1 1 7
;
```

Output 4.6 Comparing LIST and PUT Statements

```
1         SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB LIST2    STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB LIST2 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

1          DATA D;
2             INPUT X Y Z;
3             LIST;
4             IF X=Y THEN PUT 'X EQUALS Y ' X=;
5             CARDS;


RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

6          1 2 3
X EQUALS Y X=2
7          2 2 4
X EQUALS Y X=1
8          1 1 7
NOTE: DATA SET WORK.D HAS 3 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.97 SECONDS AND 372K.

9          ;
NOTE: SAS USED 372K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The current value of the SAS system option C48/C60/C96 determines how characters in the data lines are printed. When C48 or C60 is in effect, all characters in the 48- or 60-character set are printed as is. If any unprintable characters are present in the data line, SAS prints the hex representation of the line. When the system option C96 is in effect, lowercase letters are considered printable characters and are printed as they appear rather than with their hex representations. See the OPTIONS statement in "SAS Statements Used Anywhere" for more information on SAS system options.

LOSTCARD Statement

Operating systems: All

The LOSTCARD statement is used to resynchronize the input data when the SAS System encounters a missing record in data with multiple records per observation. That is, when each observation consists of several data lines, the LOSTCARD statement prevents the SAS System from reading lines from the following observation as part of the current observation when the current observation has fewer data lines than expected. Without specific instructions such as the LOSTCARD statement, SAS does not discover that a data line is missing until it reaches the end of the data. The values for the observations in the SAS data set after the missing data line was encountered may be incorrect.

The form of the LOSTCARD statement is

LOSTCARD;

You must specify the condition that indicates a lost data line as the IF condition in an IF-THEN statement. Use the LOSTCARD statement as the THEN clause of the statement or as part of a DO group following THEN.

The LOSTCARD statement is most useful when input data have a fixed number of lines per observation and when each data line for an observation contains an identification variable with the same value. See "DATA Step Applications" for information on reading data with a variable number of lines per observation.

When a LOSTCARD statement is executed, SAS takes the following steps:

1. SAS prints a lost card message, a ruler, and all the data lines read in attempting to build the current observation on the SAS log.
2. SAS does not output an observation and does not increment the automatic variable _N_. Instead, SAS discards the first data line in the group and returns to the beginning of the DATA step.
3. SAS attempts to build an observation beginning with the second data line in the group and reading the number of lines specified in the INPUT statement.
4. If the IF condition for a lost card is still true, SAS repeats steps 1-3. To make the log easier to read, SAS prints the message and ruler only once for a given group of data lines. In addition, SAS prints each data line only once; it does not repeat a data line when it is used in successive attempts to build an observation. See the second example below for an illustration.
5. When SAS encounters a group of data lines for which the IF condition is not true, SAS builds an observation, outputs it to the SAS data set, and increments the value of the automatic variable _N_.

Here is an example:

```
DATA INSPECT;
   INPUT ID 1-3 REJECT 8-10 #2 IDCHECK 1-3 PASS;
   IF ID¬=IDCHECK THEN DO;
      PUT 'ERROR IN DATA LINES ' ID= IDCHECK=;
      LOSTCARD;
      END;
   CARDS;
301    32
301    61432
```

```
302    53
302    83171
400    92845
411    46
411    99551
;
PROC PRINT;
   TITLE 'TWO DATA LINES PER OBSERVATION';
```

In this example, two input data lines make up each observation. Columns 1-3 of each line contain an identification number. You know that when the identification number in data line 1 (variable ID) does not equal the identification number in data line 2 (IDCHECK), a data line has been misplaced or left out. You specify that if a data line is missing, SAS should print the message given in the PUT statement and execute the LOSTCARD statement. Note that the first data line for the third observation (IDCHECK=400) is missing.

Below is the SAS log for the DATA step:

Output 4.7 LOSTCARD Statement: Two Data Lines per Observation

```
1         SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB LOSTCARD STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB LOSTCARD HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

1          DATA INSPECT;
2             INPUT ID 1-3 REJECT 8-10 #2 IDCHECK 1-3 PASS;
3             IF ID-=IDCHECK THEN DO;
4                PUT 'ERROR IN DATA LINES ' ID= IDCHECK=;
5                LOSTCARD;
6                END;
7             CARDS;

ERROR IN DATA LINES ID=400 IDCHECK=411
NOTE: LOST CARD.

RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

12        400    92845
13        411    46
NOTE: DATA SET WORK.INSPECT HAS 3 OBSERVATIONS AND 4 VARIABLES. 529 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.98 SECONDS AND 372K.

15         ;
16         PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 0.89 SECONDS AND 472K AND PRINTED PAGE 1.
NOTE: SAS USED 472K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

When SAS attempts to build an observation from the fifth and sixth input lines, the value read as ID (400) and the value read as IDCHECK (411) do not match. Therefore, SAS displays the lines read in attempting to build that observation, discards the first line in the group, and attempts to build an observation beginning with the second line. Since ID and IDCHECK are equal for the sixth and seventh data lines, SAS builds an observation from those lines. The resulting data set has three observations with ID values 301, 302, and 411. There is no observation for ID=400.

The output of the PRINT procedure is

Output 4.8 Data Set Produced When LOSTCARD Executed

```
                          TWO DATA LINES PER OBSERVATION                    1

                OBS    ID      REJECT    IDCHECK    PASS

                 1     301       32        301      61432
                 2     302       53        302      83171
                 3     411       46        411      99551
```

In the example below, the DATA step reads three data lines per observation. The first observation has two missing records; the second has one; and the fourth has two. The only complete observations are the third and fifth observations.

```
DATA A;
   INPUT ID X $ #2 ID2 Y $ #3 ID3 Z $;
   IF ID¬=ID2 OR ID2¬=ID3 THEN LOSTCARD;
   CARDS;
101 A
102 B
102 B
103 C
103 C
103 C
104 D
105 E
105 E
105 E
;
PROC PRINT;
   TITLE 'THREE DATA LINES PER OBSERVATION';
```

Here is the log from the DATA step:

Output 4.9 LOSTCARD Statement: Three Data Lines per Observation

```
1          SAS(R) LOG   OS SAS 5.XX           MVS/XA JOB LOSTCARD STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB LOSTCARD HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. (     ) (XXXXXXXX).

1           DATA A;
2              INPUT ID X $ #2 ID2 Y $ #3 ID3 Z $;
3              IF ID¬=ID2 OR ID2¬=ID3 THEN LOSTCARD;
4              CARDS;

NOTE: LOST CARD.

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

5          101 A
6          102 B
7          102 B
8          103 C
9          103 C
NOTE: LOST CARD.
11         104 D
12         105 E
13         105 E
NOTE: DATA SET WORK.A HAS 2 OBSERVATIONS AND 6 VARIABLES. 366 OBS/TRK.
```

(continued on next page)

```
(continued from previous page)

NOTE: THE DATA STATEMENT USED 1.00 SECONDS AND 372K.

15          ;
16          PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 0.87 SECONDS AND 472K AND PRINTED PAGE 1.
NOTE: SAS USED 472K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The five lines displayed after the first lost card message are the lines read in the first three attempts to build an observation. Note that the series of lines displayed looks like the series of lines in the data. This feature is helpful if you need to find those lines within the data. The three lines displayed after the second message are the lines read in attempting to build an observation from the seventh, eighth, and ninth data lines.

The resulting data set A has two observations:

Output 4.10 Data Set Produced When LOSTCARD Executed

```
                        THREE DATA LINES PER OBSERVATION                    1

OBS    ID      X    ID2     Y    ID3     Z

 1     103     C    103     C    103     C
 2     105     E    105     E    105     E
```

MERGE Statement

Operating systems: All

Introduction

The MERGE statement joins observations from two or more SAS data sets into single observations in a new SAS data set. The way the SAS System joins the observations depends on whether a BY statement accompanies the MERGE statement.

The form of the MERGE statement is

MERGE *SASdataset* [(*dsoptions* **IN**=*name*)]
SASdataset[(*dsoptions* **IN**=*name*)]...
[**END**=*name*];

These terms and options can appear in the MERGE statement:

SASdataset
names two or more existing SAS data sets from which to read observations each time the MERGE statement is executed. Up to 50 data set names can appear in the MERGE statement. Here are examples of valid MERGE statements:

```
MERGE MALES FEMALES;
MERGE IN.FITNESS LIB.HEALTH;
MERGE YEAR1 YEAR2 YEAR3;
MERGE TRACK SAVE.FIELD SWIM;
```

See "SAS Files" for more information on using SAS data set names.

dsoptions
specifies any number of SAS data set options in parentheses after each SAS data set name. These options include those described in "SAS Files," as well as the following special data set option unique to the SET, MERGE, and UPDATE statements:

IN=*name*
creates a variable with the name given after the equal sign. Within the DATA step, the value of the variable is 1 if the data set contributed data to the current observation and 0 otherwise. You can associate an IN= variable with each data set as shown below:

```
DATA THREE;
   MERGE ONE(IN=INONE) TWO(IN=INTWO);
```

INONE has the value 1 when data set ONE contributes information to the current observation; otherwise, it has the value 0. The value of INTWO is determined similarly. Both variables INONE and INTWO are equal to 1 if both data sets contribute information to the new observation. The IN= variable is not added to any SAS data set being created.

Operating systems: All

END=*name*
creates a variable with the name given after the equal sign to contain an end-of-file indication. The variable, which is initialized to zero, is set to 1 when the MERGE statement is processing the last observation. If the input data sets have different numbers of observations, the END= variable is set to 1 when the last observation from the data set with the most observations is being processed. The END= variable is not added to any SAS data set being created.

Operating systems: All

Naming a variable from an input data set used in a merge operation in a RETAIN statement has no effect.

The MERGE statement is flexible and has a variety of uses in SAS programming. The examples in this description illustrate basic uses of the MERGE statement. For more advanced examples, including using more than one BY variable, merging more than two data sets, and merging a few observations with all observations in another data set, see "DATA Step Applications."

Merging without a BY Statement: One-to-One Merging

When no BY statement is used, the MERGE statement joins the first observation in one data set with the first observation in another, the second observation in the data set with the second observation in another, and so forth. The number of observations in the new data set is the maximum number of observations in any of the data sets listed in the MERGE statement. When a data set being merged runs out of observations, missing values for its variables are joined with the remaining observations from the other data sets.

If a variable occurs in more than one of the data sets being merged, only one variable of that name occurs in the new data set. The value of the variable is the value in the data set listed latest (rightmost) in the MERGE statement that contains the variable and is still contributing observations.

One-to-one merge: example 1 You have two data sets, each with the same number of observations but containing different variables. One contains the name and hometown of car owners; the other contains the year and model of their cars:

```
DATA DRIVER;
   INPUT NAME $ CITY : $10.;
   CARDS;
CATHY PORTLAND
NANCY RALEIGH
SUE NASHVILLE
;
DATA VEHICLE;
   INPUT YEAR MODEL $;
   CARDS;
```

```
1985 SEDAN
1952 JEEP
1978 BUS
;
PROC PRINT DATA=DRIVER;
   TITLE 'DATA SET DRIVER';
PROC PRINT DATA=VEHICLE;
   TITLE 'DATA SET VEHICLE';
```

Output 4.11 Input Data Sets for One-to-One Merge

DATA SET DRIVER 1

OBS	NAME	CITY
1	CATHY	PORTLAND
2	NANCY	RALEIGH
3	SUE	NASHVILLE

DATA SET VEHICLE 2

OBS	YEAR	MODEL
1	1985	SEDAN
2	1952	JEEP
3	1978	BUS

You want to merge the first observation in the DRIVER data set with the first observation in the VEHICLE data set; the second observation with the second observation, and so on. The new data set contains the same number of observations as each of the input data sets, but the number of variables equals the total of the variables in the two input data sets.

```
DATA MATCH;
   MERGE DRIVER VEHICLE;
PROC PRINT;
   TITLE 'DATA SET MATCH';
```

Output 4.12 Output Data Set Produced by One-to-One Merge

DATA SET MATCH 3

OBS	NAME	CITY	YEAR	MODEL
1	CATHY	PORTLAND	1985	SEDAN
2	NANCY	RALEIGH	1952	JEEP
3	SUE	NASHVILLE	1978	BUS

Merging with a BY Statement: Match-Merging

If you want to match observations from two or more SAS data sets based on the values of some variables, then use a BY statement after the MERGE statement. In order to perform match-merging, at least one variable must be common to all data sets, and each data set must be sorted by these variables. (See "DATA Step Applications" for a match-merging technique you can use when all the data sets

do not share a common variable.) The variables used for matching are called BY variables; the BY statement is used to identify the matching variables.

Nonmatches When nonmatching BY values occur, SAS processes all observations with the lower BY value before processing any observations with a higher BY value. The FIRST. and LAST. variables are used to detect the beginning and end of BY groups, and thus let you control whether to output, delete, or count multiple observations in a BY group. The IN= variable, described earlier, lets you know if a data set contributed information to the observation being built; thus, you can use it to detect nonmatches.

Multiple observations with the same BY value in a data set The match-merge operation combines all the data from each data set that has an observation with the current BY values. That is, if a data set has more than one observation with the same BY value, the match-merge operation outputs each observation. The first observation of the BY group is combined with the first observation in the BY group from every data set with observations for that BY value; the second observation is combined with the second, and so on. The resulting data set contains as many observations for a BY group as the largest number in that BY group in any of the data sets. The total number of observations in the new data set is the sum of the largest number of observations in each BY group across all input data sets.

When an input data set exhausts the observations in a BY group, the values from the last observation in that BY group are retained and merged with the remaining observations in that BY group from other data sets. The value of 1 that was assigned to the IN= variable when the data set began to contribute information in that BY group is also retained. You can reset the IN= variable to 0 in an assignment statement before the MERGE statement if you want to detect only new information in the BY group.

If one input data set contains no observations in a particular BY group, that data set contributes missing values to its unique variables in those observations in the new data set. The value of the IN= variable is 0 for that data set throughout that BY group.

Variables with the same name in more than one data set If a variable other than a BY variable occurs in more than one data set being merged, only one variable of that name occurs in the new data set. When a BY group has only one observation in each of the data sets being joined, the value of the variable in the new observation is the value from the data set mentioned latest in the MERGE statement. When multiple observations occur within a BY group, the value in the new data set is the value from the data set mentioned latest in the MERGE statement that is still contributing information to the BY group.

Match-merge: example 2 You have two data sets. Data set PERSON contains the variables NAME and SEX; data set PLACE contains the variables NAME, CITY, and REGION.

```
DATA PERSON;
   INPUT NAME $ SEX $;
   CARDS;
MARY F
ANN F
TOM M
;
PROC PRINT;
```

```
   TITLE 'DATA SET PERSON';
DATA PLACE;
   INPUT NAME $ CITY $ REGION;
   CARDS;
JOSE ERIE 5
MARY MIAMI 2
MARY TAMPA 7
ANN TAMPA 6
;
PROC PRINT;
   TITLE 'DATA SET PLACE';
```

Output 4.13 Input Data Sets for Match-Merge

DATA SET PERSON 1

OBS	NAME	SEX
1	MARY	F
2	ANN	F
3	TOM	M

DATA SET PLACE 2

OBS	NAME	CITY	REGION
1	JOSE	ERIE	5
2	MARY	MIAMI	2
3	MARY	TAMPA	7
4	ANN	TAMPA	6

You want to merge each observation in data set PERSON with the observation in PLACE that has a matching value of the common variable NAME. These steps perform the match-merge operation:

```
PROC SORT DATA=PERSON;
   BY NAME;
PROC SORT DATA=PLACE;
   BY NAME;
DATA RESULT;
   MERGE PERSON PLACE;
   BY NAME;
PROC PRINT;
   TITLE 'DATA SET RESULT';
```

Output 4.14 Output Data Set Produced by Match-Merge

DATA SET RESULT 3

OBS	NAME	SEX	CITY	REGION
1	ANN	F	TAMPA	6
2	JOSE		ERIE	5
3	MARY	F	MIAMI	2
4	MARY	F	TAMPA	7
5	TOM	M		.

Since data sets PERSON and PLACE are not in sorted order of the variable NAME, you must sort them before merging them. The BY group with the lowest BY value, ANN, has one observation in data set PERSON and data set PLACE, so data set RESULT contains one ANN observation with the information from the ANN observation in data sets PERSON and PLACE. The BY group with the next BY value, JOSE, has an observation only in PLACE, so in data set RESULT the JOSE observation has a missing value for SEX, the variable from data set PERSON. The next BY value, MARY, has one observation in PERSON and two observations in PLACE. Thus, the sex value of 'F' from the MARY observation in PERSON is combined with both MARY observations in PLACE to produce two observations with sex values of 'F' in RESULT. The final BY value, TOM, exists only in PERSON, so the CITY and REGION values from PLACE are missing in the TOM observation in RESULT.

Match-merge for table lookup: example 3 You have one data set that is your table; it contains an identifier variable and corresponding descriptions. You want to merge the descriptions with another data set that contains the identifier variable. For example, say the data set containing the table has two variables, NUMBER and DESCRIPT. NUMBER contains the part number and DESCRIPT contains the part description. The other data set contains the part number and the name of a customer. Both data sets are sorted by NUMBER, the identifier variable.

```
DATA PARTDATA;
   *THE TABLE;
   INPUT NUMBER DESCRIPT $12.;
   CARDS;
155 SCREWDRIVER
244 WRENCH
501 PLIERS
796 HAMMER
;
PROC SORT DATA=PARTDATA;
   BY NUMBER;
PROC PRINT DATA=PARTDATA;
   TITLE 'PARTDATA--THE ORDER FILE';
```

Output 4.15 Table Data Set for Table Lookup

PARTDATA--THE ORDER FILE 1

OBS	NUMBER	DESCRIPT
1	155	SCREWDRIVER
2	244	WRENCH
3	501	PLIERS
4	796	HAMMER

```
DATA ORDERS;
   *THE LIST OF PARTS THAT WERE ORDERED AND THE CUSTOMERS
      WHO ORDERED THEM;
   INPUT NUMBER NAME & $16.;
   CARDS;
155  R. B. HOADLEY
```

```
155  G. C. FREE
244  A. T. CANNON
244  M. C. WHITE
244  Z. A. DE LA CRUZ
796  S. C. FOXX
796  M. C. WHITE
;
PROC SORT DATA=ORDERS;
   BY NUMBER;
PROC PRINT DATA=ORDERS;
   TITLE 'ORDERS--LIST OF NAMES AND ORDERS';
```

Output 4.16 List of Names and Orders to be Looked Up

```
                ORDERS--LIST OF NAMES AND ORDERS                    2

               OBS    NUMBER    NAME

                1       155     R. B. HOADLEY
                2       155     G. C. FREE
                3       244     A. T. CANNON
                4       244     M. C. WHITE
                5       244     Z. A. DE LA CRUZ
                6       796     S. C. FOXX
                7       796     M. C. WHITE
```

```
DATA COMPLETE;
   MERGE PARTDATA ORDERS;
   BY NUMBER;
PROC PRINT;
   TITLE 'COMPLETE';
```

Output 4.17 Complete Merged Data Set for Table Lookup

```
                              COMPLETE                              3

          OBS    NUMBER     DESCRIPT      NAME

           1       155     SCREWDRIVER   R. B. HOADLEY
           2       155     SCREWDRIVER   G. C. FREE
           3       244     WRENCH        A. T. CANNON
           4       244     WRENCH        M. C. WHITE
           5       244     WRENCH        Z. A. DE LA CRUZ
           6       501     PLIERS
           7       796     HAMMER        S. C. FOXX
           8       796     HAMMER        M. C. WHITE
```

Since you combined the data sets in order to have a list of customers with a description of the part they buy, you want to delete observations from the table (PARTDATA) that have no match in ORDERS. Part number 501, PLIERS, has no match in ORDERS. You can use the IN= variable on the ORDERS data set to ensure that all observations in the result are also in the ORDERS data set:

```
DATA COMPLETE;
   MERGE PARTDATA ORDERS(IN=A);
   BY NUMBER;
   IF A;
```

```
PROC PRINT;
   TITLE 'COMPLETE (WITH IN=)';
```

Observations are only output to COMPLETE when the value of the IN variable A is 1:

Output 4.18 Merged Data Set Showing Only Parts Ordered

COMPLETE (WITH IN=) 4

OBS	NUMBER	DESCRIPT	NAME
1	155	SCREWDRIVER	R. B. HOADLEY
2	155	SCREWDRIVER	G. C. FREE
3	244	WRENCH	A. T. CANNON
4	244	WRENCH	M. C. WHITE
5	244	WRENCH	Z. A. DE LA CRUZ
6	796	HAMMER	S. C. FOXX
7	796	HAMMER	M. C. WHITE

Comparison of match-merge and one-to-one merge When all data sets being merged contain the same number of observations in each BY group, the result of a match-merge operation is the same as that of a one-to-one merge. However, if any input data set contains a different number of observations in a BY group (such as two observations with the BY variable value ID=3001 and no observation with ID=3002), a match-merge operation outputs all observations in the BY group for all data sets. You can detect the discrepancy in the number of observations in the BY group either with FIRST. and LAST. variables or by noting an unexpected number of observations in the output data set. In the same case a one-to-one merge operation, which joins observations sequentially, causes subsequent observations to become mismatched. You cannot detect the mismatch except by observing unusual combinations of values in the output data set (such as NAME='MARY' and SEX='M').

MISSING Statement

Operating systems: All

You can use a MISSING statement to declare that certain values in your input data represent special missing values for numeric data. Although you can use a MISSING statement anywhere in a SAS program, it is most useful in the DATA step.

A MISSING statement has this form:

MISSING *values;*

where

values are the values in your input data that you are using to represent special missing values. These special missing values may be any of the twenty-six capital letters of the alphabet (not lowercase letters) or the underscore (_). See "Missing Values" for further discussion of special missing values in the SAS System.

For example, with survey data, you want to identify certain kinds of missing data. Suppose an 'A' is coded when the respondent was absent from home at the time of the survey; an 'R' when the respondent refused to answer.

```
DATA SURV;
   MISSING A R;
   INPUT ID ANSWER1;
   CARDS;
1001 2
1002 R
1003 1
1004 A
1005 2
more data lines
;
```

The MISSING statement indicates that values of A and R in the input data lines are to be considered special missing values rather than invalid numeric data values.

Null Statement

Operation systems: All

The null statement is a single semicolon. The statement does not perform any action but can play the role of a placeholder. Although a null statement may be used anywhere in a SAS program, it is most useful in the DATA step. For example, in some SAS programs that include a CARDS statement you may also need a null statement to signal the end of the data lines.

The form of a null statement is

```
;
```

In the DATA step, a CARDS statement signals to the SAS System that data lines follow immediately in the job stream. SAS recognizes the end of the data lines when it sees a semicolon on a line. If the first line after the last data line already contains a semicolon, you do not need a null statement. For example,

```
DATA COMM;
   INPUT X Y Z;
   CARDS;
data lines
PROC PRINT;
```

However, if a semicolon does not appear in the line after the last data line, a null statement can signal the end of the data:

```
DATA COMM;
   INPUT X Y Z;
   CARDS;
data lines
;
PROC PRINT
     DATA=COMM (KEEP=X RENAME=(X=VISIT1));
```

When your data contain semicolons and you use the CARDS4; statement, the null statement is indicated by four semicolons. For example,

```
DATA COMM;
   INPUT X Y Z;
   CARDS4;
data lines containing semicolons
;;;;
```

Although no action is performed by the statement, it is considered an executable statement. Thus, a label can precede it. For example,

```
DATA LAB;
   INFILE IN;
   INPUT X Y Z;
   IF X=. THEN GO TO FIND;
   LIST;
FIND: ;
   DROP X;
```

See the section **Labels, Statement** listed alphabetically in this chapter for more information on using statement labels.

OUTPUT Statement

Operating systems: All

The OUTPUT statement tells the SAS System to write the current observation to the data set being created. The form of the OUTPUT statement is

OUTPUT [*SASdataset*] **...;**

where

SASdataset
: optionally specifies the data sets to which the current observation should be written. More than one SAS data set name can be given. All names specified must also appear in the DATA statement. When no SAS data set name is given, the current observation is written to all data sets being created in the step.

Simple SAS DATA steps do not need an OUTPUT statement since observations are automatically output before SAS returns to the beginning of the step for another execution. The OUTPUT statement is useful when you need to control the normal output of observations in situations like these:

- you want to create two or more observations from each line of input data
- you are creating more than one SAS data set from one input data file
- you want to combine several input observations into one observation.

When an OUTPUT statement appears among the program statements in the DATA step, SAS adds an observation to the SAS data set(s) only when the OUTPUT statement is executed. No automatic output occurs.

Creating several observations from one input line Here is an example of creating several observations from one input line. Each line contains a subject identifier and three measurements for that subject. For each input line, you want to produce three observations. Each new observation should contain the subject identifier and one measurement.

```
DATA REPEAT;
   INPUT SUBJECT $ MEASURE1-MEASURE3;
   DROP MEASURE1-MEASURE3;
   MEASURE=MEASURE1; OUTPUT;
   MEASURE=MEASURE2; OUTPUT;
   MEASURE=MEASURE3; OUTPUT;
   CARDS;
A 2 5 4
B 3 6 2
;
```

The new data set contains the observations shown in **Output 4.19**.

Output 4.19 Creating Several Observations from One Input Line

REPEAT 1

OBS	SUBJECT	MEASURE
1	A	2
2	A	5
3	A	4
4	B	3
5	B	6
6	B	2

Creating more than one data set in a single DATA step These statements create two data sets from a single input record:

```
DATA COLLEGE HISCHOOL;
   INPUT NAME $ 1-30 SEX $ YRS_EDUC;
   IF YRS_EDUC<=12
      THEN OUTPUT HISCHOOL;
   ELSE OUTPUT COLLEGE;
   CARDS;
data lines
;
```

The data set HISCHOOL contains all observations with a YRS_EDUC value of 12 or less. Data set COLLEGE contains observations with a YRS_EDUC value greater than 12.

Combining information from several records These statements combine the information from several input records into one observation in the SAS data set:

```
PROC SORT DATA=PAYROLL; BY SSN;
DATA CHECKS;
   SET PAYROLL; BY SSN;
   IF FIRST.SSN THEN TOT_PAY=0;
   TOT_PAY+PAY;
   DROP PAY;
   IF LAST.SSN THEN OUTPUT;
```

SAS data set PAYROLL, which has been sorted by SSN, is used as input to the DATA step. PAYROLL contains several observations for each SSN. Since the BY statement appears in the DATA step, the FIRST. and LAST. automatic variables can be used to check for the first and last observations with each SSN value. A sum statement accumulates total pay for each SSN. When an observation is the last with a particular SSN value, SAS writes the observation to the new data set. Thus, the new data set contains one observation for each SSN value in the PAYROLL data set.

The contents of data set PAYROLL after sorting are shown in **Output 4.20**.

Output 4.20 Contents of Data Set PAYROLL

PAYROLL AFTER SORTING 2

OBS	SSN	PAY
1	111442222	100.00
2	111442222	25.00
3	333115555	160.00
4	333115555	80.00
5	777668888	142.66

The contents of data set CHECKS are shown in **Output 4.21**.

Output 4.21 Contents of Data Set CHECKS

CHECKS 3

OBS	SSN	TOT_PAY
1	111442222	125.00
2	333115555	240.00
3	777668888	142.66

PUT Statement

Operating systems: All

Introduction

The PUT statement writes lines to the SAS log, to the SAS procedure output file, to punched cards, or to any file that can be specified in a FILE statement. The file specified by the most recently executed FILE statement is the *current output file*.

If no FILE statement executes before a PUT statement in the current execution of a DATA step, the lines are written on the SAS log. See the **FILE statement** description earlier in this chapter for more information.

The PUT statement writes lines containing variable values or strings of text. Variable values can be labeled with the name of the variable by using named output. With specifications in the PUT statement, you list items to be written and describe their format.

You can write variable values in one of three basic output styles: column, list or free-form, or formatted style. Simple examples showing how to specify these output styles are given below. Each style is fully described in a later section.

With **column style** enter a range of numbers after the variable name; these numbers specify a range of columns into which values are written, for example,

```
PUT NAME 6-15 WEIGHT 17-19;
```

With **list style** simply list the variables in the PUT statement in the order you want to write them:

```
PUT NAME WEIGHT SEX;
```

With **formatted style** specify a *format* after the variable name:

```
PUT DATE MMDDYY8. TIME HHMM5.;
```

The general form of the PUT statement is

PUT [*specification*] ...;

where *specification* describes how a variable's value or a text string is written in the output line. You can use the following specifications in the PUT statement:

variable
names the variable whose value is to be written

'characterstring'
specifies a string of text to be written by the PUT statement. The string must be enclosed in quotes. See **Writing Character Constants** below.

pointercontrol
moves the output pointer to a specified line or column. See **Pointer Controls and Format Modifiers** below.

INFILE
writes the last line read either from the current input file or from lines following a CARDS statement. See **Writing the Current Input Line** below.

ALL
writes the values of all variables, including _ERROR_ and _N_, defined in the current DATA step using named output. See **Listing Values of the Current Variables** below.

You can combine many of these features in a single PUT statement. The PUT statement writes each item in the order specified, for example,

```
PUT NAME 'WEIGHS ' WEIGHT 17-19 ' ON ' DATE MMDDYY8;
```

A PUT statement with no specifications (a null PUT statement), for example,

```
PUT;
```

causes the current output line to be immediately written to the current file, even if the current output line is blank. The null PUT statement releases an output line being held by a previous PUT statement with a trailing @. See **Pointer Controls and Format Modifiers** for more information on the trailing @.

Column Style

Column style describes the output lines by giving the variable's name and the columns its value is to occupy in the output line. The PUT statement writes the values of the variable in the specified columns of the output line.

If the values require fewer columns than specified, character variables are left aligned in the specified columns, and numeric variables are right aligned in the specified columns.

The form of the PUT statement for writing the value of one variable with column output is

PUT [*variable*] [=] [$] *startcolumn*[-*endcolumn*] [.*decimalplaces*];

where

variable
names the variable whose value is to be written.

=
specifies that the value is to be labeled with the variable name and an equal sign. See **Labeling Variable Values: Named Output** for more information.

$
: indicates that the variable contains character values rather than numeric values. If the variable has already been defined as a character variable (for example, in an INPUT statement), you can omit the $ sign.

startcolumn
: is the first column of the field where the value is to be written in the output line.

-endcolumn
: is the last column of the field for the value. If the value is to occupy only one column in the output line, omit the *-endcolumn* specification.

.decimalplaces
: is a period followed by a positive integer that specifies the number of digits you want on the right side of the decimal point.[1]

There is no limit to the number of column style specifications you can specify in a single PUT statement. This PUT statement writes the values of two variables on the current output line:

```
PUT NAME $ 1-8 ADDRESS $ 10-35;
```

The value of NAME is written in columns 1 through 8; then the value of ADDRESS is written in columns 10 through 35 on the same output line.

When you list more than one item, the PUT statement writes each item in the order specified. For example, the statement

```
PUT FIRST 73-80 SECOND 10-12;
```

first writes a value of the variable FIRST in columns 73-80, then writes a value of SECOND in columns 10 through 12 on the same output line.

The specified range of columns must provide sufficient space for the value. If you are using named output, for example, remember that the variable name, the equal sign (=), and the value must fit in the columns specified. (See **Labeling Variable Values: Named Output** below.)

If a variable is previously defined as character by a statement earlier in the step, a $ is not necessary in the PUT statement. For example, in the DATA step,

```
DATA A;
   INPUT NAME $ 1-15;
   FILE OUT;
   PUT NAME 1-15;
```

no $ is necessary in the PUT statement since the variable NAME is defined as character in the INPUT statement.

List Style

With list style you simply list the names of the variables you want written. The PUT statement writes the value of each variable listed in the PUT statement, leaves a blank, and then writes the next value.

The form of the PUT statement for writing one variable with list output is

PUT *variable*[**=**] [**$**];

where

variable names the variable you want written.

= specifies that the variable value be written using named output. See **Labeling Variable Values: Named Output** for more information.

$ specifies that the variable is a character variable. The $ is not necessary if the variable has been previously defined as character.

In the following DATA step the PUT statement writes values of NAME, SEX, and AGE using list output style:

```
DATA CLASS;
   INPUT NAME $ 1-10 SEX $ 12 AGE 14-15;
   PUT NAME SEX AGE;
   CARDS;
HENRY       M  13
JOE         M  14
HENRIETTA   F  11
;
```

The following data lines are written to the SAS log by the PUT statement:

```
HENRY M 13
JOE M 14
HENRIETTA F 11
```

Notice that one blank separates the data values on the output line.

With list output, missing values for numeric variables are written as a single period (.). Character values are left aligned in the field; leading and trailing blanks are removed. Therefore, if you want blanks included when the value is written (in addition to the blank inserted after each value), use formatted instead of list style.

You can use the list output form and also specify the format to be used for writing the value if you include the colon (:) format modifier. See **Pointer Controls and Format Modifiers** below.

Formatted Style

Formatted style describes the output lines by listing variable names and formats for writing the values. With formatted style, the PUT statement writes each value using the format that follows the variable name. No blanks are automatically added between values. Formatted style combined with pointer controls discussed in **Pointer Controls and Format Modifiers** make it possible to specify the exact line and column location to write each variable.

The form of the PUT statement for writing one variable with formatted output is

PUT *variable*[=] *format.*;

where

variable names the variable you want to write.

= specifies that the variable value should be written using named output. See **Labeling Variable Values: Named Output** for more information.

format. specifies a format to use when writing the data values. The format can be either a SAS format (see "SAS Informats and Formats") or a format you define (see the FORMAT procedure description). Every format specification **must** include a period.

You can list several variables by using a format list:

PUT (*variablelist*) (*formatlist*) **... ;**

where

(*variablelist*)
: is any valid variable list enclosed in parentheses.

(*formatlist*)
: lists the formats to be used to write the preceding list of variables. See **Grouping Variables and Formats** below.

The width you specify for the format must provide enough space to write the value and any commas, dollar signs, or other special characters that the format includes.

Suppose the value of X is 100. You want to write the value using the DOLLAR. format and include two decimal places for cents. This PUT statement,

```
PUT X DOLLAR7.2;
```

writes the formatted value, which takes 7 columns:

```
$100.00
```

If the value uses fewer columns than specified, character values are left aligned and numeric values are right aligned by default in the field specified by the format width.[2]

Pointer Controls and Format Modifiers

Most of the PUT statements shown so far are examples of simple column, list, and formatted styles. Two advanced features—pointer controls and format modifiers—can add flexibility regardless of the output style you are using.

SAS keeps track of its position on each output line with a pointer. With specifications in the PUT statement, you can control pointer movement from column to column and line to line. For example, if each record requires several lines and you are writing a value that begins in the tenth column of the second line, the pointer value indicates that the current line is 2 and the current column, 10.

The pointer controls and format modifiers are described below:

Pointer controls

@n
: moves the pointer to column *n*.

@pointvariable
: moves the pointer to the column given by the value of *pointvariable*. The *pointvariable* must be numeric.

  To move the pointer to a specific column, use the @ followed by the column number or by a variable name whose value is that column number. For example, the statement

```
PUT @15 SALES;
```

  moves the pointer to column 15 and writes the value of SALES using list output.

  The PUT statement in this example:

```
DATA ONE;
   INPUT SPACE X;
   PUT @SPACE X 5.2;
```

moves the pointer to the column indicated by the value of SPACE and writes the value of X using the 5.2 format.

The pointer can go backward as well as forward. For example, this PUT statement

```
PUT @26 BOOK @1 COMPANY;
```

first writes BOOK's value beginning in column 26 and then returns to column 1 on the same line to write COMPANY's value.

For more information see **Specifying Values for Pointer Controls** below.

+n

moves the pointer forward *n* columns.

+pointvariable

moves the pointer forward or backward the number of columns indicated by the value of *pointvariable*. The pointer moves backward if the *pointvariable* has a negative value. The *pointvariable* must be numeric.

The + pointer direction, followed by a number or a variable name, moves the pointer by the number or the variable's value. The movement is relative to where the pointer is located. For example, the statement

```
PUT @23 LENGTH 4. +5 WIDTH;
```

moves the pointer to column 23, writes LENGTH's value in four columns (23, 24, 25, and 26) leaving the pointer at column 27, then advances the pointer five columns and begins writing a WIDTH value in column 32.

For more information see **Specifying Values for Pointer Controls** below.

A + can cause the SAS System to attempt to write past the current line length. See **When the Pointer Goes Past the End of a Line** for more information.

#n

moves the pointer to line *n*.

#pointvariable

moves the pointer to the line number indicated by the value of *pointvariable*. The *pointvariable* must be numeric.

When you want to write several lines of data with one PUT statement, you can use the # to indicate on which line the information is to be written. For example, the statement

```
PUT @12 NAME $10. #2 ID 3-4;
```

writes the value for NAME beginning in column 12 of the first output line and then writes a value for ID in columns 3 and 4 of the second line.

The following statement:

```
PUT #4 ID #2 NAME;
```

writes a value for ID on line 4 and moves back to line 2 to write the NAME value.

If the N= option of the FILE statement is not specified, the highest value used with the # pointer control (*n* or *pointvariable*) in the current DATA step is the default value for N=. The N= option is described with the FILE statement.

For more information see **Specifying Values for Pointer Controls** below.

/

moves the pointer to column 1 of the next line. For example, the statement

```
PUT AGE GRADE / SCORE1-SCORE5;
```

first writes values of AGE and GRADE on one line and then skips to the next line to write values of SCORE1-SCORE5 beginning in column 1.

@, trailing

holds a data line for another PUT statement. The @ is called a *trailing at-sign* because it must follow all other items in the PUT statement.

Usually, each PUT statement in a DATA step writes a new data line. When you want to use more than one PUT statement to write values on the same output line, you can use an @ as the last item in your PUT statement to hold the pointer at its current location. The next PUT statement executed in that DATA step then writes to the same line rather than to a new line.

For example,

```
DATA _NULL_;
   INPUT NAME $ WEIGHT;
   PUT NAME @;
   IF WEIGHT¬=. THEN PUT @15 WEIGHT @;
   PUT;
```

The trailing @ in the first PUT statement holds the current output line after NAME is written; thus, if the value of WEIGHT is not missing, a WEIGHT value is written on the line with NAME. The trailing @ in the second PUT statement holds the line after WEIGHT is written. The final null PUT statement always releases the current line and positions the pointer at column one on the next line.

Suppose you have a SAS data set named DEXT containing dexterity tests scores for a group of children. Each observation in the data set contains three variables: CHILD, a child's name; SCORE, a dexterity score; and TYPE, the hand (right or left) for which the score was recorded. There are two observations for each child—one for the right-hand score and one for the left-hand score. You want to print the data with the right- and left-hand scores on the same output line:

```
DATA _NULL_;
   SET DEXT;
   BY CHILD;
   IF FIRST.CHILD THEN PUT CHILD @;
   IF TYPE='LEFT' THEN PUT @25 SCORE 4.2 @;
   ELSE IF TYPE='RIGHT' THEN PUT @35 SCORE 4.2 @;
   IF LAST.CHILD THEN PUT;
```

In this example, if the observation is the first for CHILD, the child's name is written. The pointer holds that line until the left- or right-hand score is written. Still holding the line, the next record is read from DEXT and the score written; since the record is the last for a child, the line is released by the last PUT statement.

Note: the trailing @ specified in the PUT statement holds the current line for the next execution of a PUT statement even if

another execution of the DATA step begins. Thus, the double trailing @ pointer control, which is required in the INPUT statement to hold the current input line, is not needed in the PUT statement.

A trailing @ holds the current output line until one of the following is encountered:

- a PUT statement without a trailing @, as shown above
- a PUT statement specifying _PAGE_ (described below)
- the end of the current line (as specified by the current LINESIZE= value)
- the end of the data.

Using a trailing @ can cause the SAS System to attempt to write past the current line length, since the pointer value is unchanged when the next PUT statement executes. See **When the Pointer Goes Past the End of a Line** for more information.

PAGE

advances the pointer to the first line of a new page. The SAS System automatically begins a new page when a line exceeds the current PAGESIZE value.

If the current output file is a print file, _PAGE_ produces an output line containing the carriage control character one (1).

PAGE is unnecessary in PUT statements executed by a HEADER= option in the FILE statement. Lines produced by those statements are automatically printed on a new page.

For example, suppose you want to go to the first line of a new page after printing information for the last observation in a county, as in this example:

```
DATA _NULL_;
   SET STATES;
   BY COUNTY;
   FILE PRINT;
   PUT NAME 1-10 @15 POP COMMA9. ;
   IF LAST.COUNTY THEN PUT _PAGE_;
```

PUT _PAGE_ advances the pointer to line 1 of the new page when the value of LAST.COUNTY is 1. (A discussion of FIRST. and LAST. variables is given in the BY statement description.)

You can specify the _PAGE_ option in the PUT statement along with variables, strings of text, and other PUT statement features. For example, suppose you want to print a footer message before exiting from the page:

```
DATA _NULL_;
   SET STATES;
   BY COUNTY;
   FILE PRINT;
   PUT NAME 1-10 @15 POP COMMA9. ;
   IF LAST.COUNTY THEN PUT // 'THIS IS THE LAST OF '
   COUNTY $10. _PAGE_;
```

When an observation is the last for a county, the PUT statement skips two lines and prints the message 'THIS IS THE LAST OF ' followed by the current value of COUNTY before skipping to the next page.

OVERPRINT

prints over the previous line.

You can use the OVERPRINT option when your PUT statements are directed to a print file and when the N= option of the FILE statement has a value of 1. The OVERPRINT option starts a new line with a plus (+) as the carriage control character. The specifications following the keyword OVERPRINT are also written to the new line. When the file is printed, lines with a + as the carriage control character overprint the preceding line.

For example, this PUT statement underlines a title by overprinting with underscores:

```
PUT 'TITLE OF PAGE' OVERPRINT '_____________';
```

If OVERPRINT is the first keyword in a PUT statement, the line written by that statement overprints the last line written by a previous PUT statement in the current output file. For example, the statements

```
DATA _NULL_;
   SET CLASS;
   FILE PRINT;
   PUT NAME 1-10 @15 GRADE 2.;
   IF GRADE >= 96 THEN PUT OVERPRINT @15 '__';
```

underline grades above 95 on the output line.

You can use the OVERPRINT option in a PUT statement with other pointer controls:

```
PUT @5 NAME $8. OVERPRINT @5 '________'/ @20 ADDRESS;
```

This PUT statement writes a value for NAME, underlines it by overprinting underscores, then goes to the next line to write an ADDRESS value.

Note: carriage control characters are interpreted by the operating system when files are printed. A plus sign (+) is the standard carriage control character for producing overprinted output on a line printer. Some output devices are incapable of overprinting, and others may assign a different meaning to the +. For example, on an IBM 6670 Document Printer a line containing a + may be printed in boldface.

The OVERPRINT option in the PUT statement has no effect when you are writing lines at a terminal.

Specifying values for pointer controls When you specify #*n* or @*n*, you move the pointer to a specific line or column. Thus, the value of *n* must be a positive integer. When you use a *pointvariable* with # and @, the value of the variable is truncated to a positive integer if necessary. Since +*n* moves the pointer forward relative to its current location, the value specified with +*n* is also a positive integer.[3]

The value for the *pointvariable* specified with + can be a negative or positive integer to allow you to move the column pointer forward or backward relative to its current location. You can move the pointer backward by assigning a negative value to a variable to be used as the *pointvariable*. For example, after writing a value with list output, you may want to back up one column to write over the blank that the PUT statement automatically writes with list output. This DATA step can be used:

```
DATA ONE;
   INPUT X Y Z;
   M=-1;
```

```
PUT X +M Y +M Z;
```

Since the X, Y, and Z values are written using list output, a blank would normally be added after each value is written. The +M pointer control moves the pointer back one space to eliminate the blank.

Format modifiers

*n** specifies that the next format in a format list is to be repeated *n* times.[4] For example,

```
PUT (GRADES1-GRADES3) (3* 7.2);
```

writes GRADES1, GRADES2, and GRADES3 with the 7.2 format.

colon (:) precedes a format and causes the SAS System to write the variable's value using the format, trimming off leading and trailing blanks and following the value with one blank. For example, the statements

```
DATA A;
   X=12353.2;
   Y=15;
   PUT X : COMMA10.2 Y : 5.2;
```

produce the line

```
12,353.20 15.00
```

Special Topics

When the pointer goes past the end of a line SAS does not write an output line that is longer than the current line length (as specified by the LINESIZE= value). You may inadvertently send the pointer beyond the current line length with one of, or some combination of, the following specifications:

- using the @ sign to hold the current line for a value that does not fit in the remaining space
- using the + pointer control with a value that moves the pointer to a column beyond the current line length
- specifying a column range that exceeds the current line length (for example, PUT X 90-100; when LINESIZE=80)
- attempting to write any item (variable value or text string) that does not fit in the space remaining on the current output line.

When a PUT statement attempts to write past the end of the current line, SAS withholds the entire item that overflows the current line, writes the current line, then writes the overflow item on a new line. This is the default action.[5]

The pointer's location after writing a data value The pointer location after a value is written depends on the output style used in the PUT statement. If list style is used, SAS sets the pointer to the second column after the value since the PUT statement automatically skips a column after writing each value. If column or formatted style is used, the pointer is set to the first column after the end of the field specified in the PUT statement.

After the PUT statement writes a character string, the pointer is located at the first column after string. After an _INFILE_ specification, the pointer is located at the next column after the record written from the current input file.

You can find the pointer's current location using the COLUMN= and LINE= options of the FILE statement. See the FILE statement description for more information.

Grouping variables and formats When you want to write values in a pattern on the output lines, using format lists can shorten your coding time. A format list consists of a list of variable names, enclosed in parentheses, followed by a corresponding list of formats, also enclosed in parentheses. You can use as many format lists as necessary in a PUT statement. You can include any of the pointer controls (@, #, /, +, and OVERPRINT) in the list of formats. However, format lists cannot be nested. You must separate items in a format list either by blanks or by commas.

The following example uses a format list to write the five variables SCORE1-SCORE5, one after another, using four columns for each value with no blanks in between:

```
PUT (SCORE1-SCORE5) (4. 4. 4. 4. 4.);
```

When there are more variables than format items, SAS uses the same format list again and again until all the variables have been written. So a simpler way to write the same PUT statement is

```
PUT (SCORE1-SCORE5) (4.);
```

When the format list includes more formats and pointer controls than are needed, the PUT statement ignores any remaining specifications in the format list after all the variable values have been written. For example, the PUT statement

```
PUT (X Y Z) (2.,+1);
```

writes the value of X using the 2. format, skips the next column, writes Y's value using the 2. format, skips a column, then writes Z's value using the 2. format. The +1 pointer control remaining in the third cycle of the format list is not used.

Writing character constants There is no limit to the number of character strings you can specify in a PUT statement. When using list output, you can specify character strings anywhere in the PUT statement. When using column or formatted output, character strings cannot be specified between a variable and its column locations or a variable and its format.

You can repeat a character constant on the output line by using the n* format modifier or by using the REPEAT function.[6]

After writing a character constant, the pointer is located at the first column after the constant. Thus, if you are planning to follow a character constant with a value on the output line, you may want to use a blank as the last character of the constant. For example, say that values for the variables YEAR and TOTAL are 1985 and 1000, respectively. The statement

```
PUT 'THE PROFIT FOR ' YEAR 'IS ' TOTAL;
```

writes the line

```
THE PROFIT FOR 1985 IS 1000
```

The blank in each of the character constants prevents the values for YEAR and TOTAL from following the constants with no intervening space.

When insufficient space remains on the current line to write the entire text of a character string, SAS withholds the entire string and takes the action described in **When the Pointer Goes Past the End of a Line** above.

Writing the current input line When _INFILE_ is specified, the PUT statement writes the last record read from the file currently being used as input. The current input file can be the file specified by the most recently executed INFILE statement or data lines following a CARDS statement. Consider the following program:

```
DATA _NULL_;
   INPUT;
   PUT _INFILE_;
CARDS;
data lines
;
```

With each execution of the DATA step, the PUT _INFILE_ statement writes the next line of data following the CARDS statement on the SAS log.

If the most recent INPUT statement for the current input file read more than one line, only the last line is written by the _INFILE_ specification.

If you are updating a VSAM file, you can use the _INFILE_ specification with a trailing @ to copy the current input record to the output buffer. See the *SAS Guide to VSAM Processing* for details.

Listing values of the current variables When _ALL_ is specified, the PUT statement writes all currently defined variables using named output. For example, the statement

```
PUT _ALL_;
```

writes labeled values of all currently defined variables including the automatic variables _ERROR_ and _N_. Named output is described below.

Labeling variable values: named output The PUT statement can label values with their variable names. This form of writing values is called *named output*. You can specify named output with column, list, or formatted output by following the variable name in the PUT statement with an equal sign (=). You can label character or numeric variables. For example, this statement

```
PUT NAME= @12 HEIGHT= WEIGHT=;
```

writes all three variables using named output. Thus, if the current record contains the values ANN, 63.3, and 95.1, the PUT statement writes the following line:

```
NAME=ANN   HEIGHT=63.3 WEIGHT=95.1
```

Formats following the equal sign specify an output format for the variable. For example, the statement

```
PUT AMOUNT=DOLLAR8.2;
```

writes the current value of AMOUNT in named output style with the format DOLLAR8.2.

Arrays You can write an array element with the PUT statement just as you can write any other SAS variable. However, you cannot use an array as a variable for pointer control; that is, you cannot use an array as a *pointvariable*. You can use both explicitly subscripted and implicitly subscripted arrays.

If you use an explicitly subscripted array, enclose the subscript in braces after the array name, as follows:

```
PUT ARR1{INDX+2} ...
```

The index can be any valid SAS expression as long as the expression evaluates to a valid subscript when the PUT statement executes.[7]

If you use an implicitly subscripted array, the index variable defined for the array (or the default index variable _I_) determines which element of the array to write. If the subscript does not evaluate to a valid value at the time the PUT statement executes, you will get an error message. Refer to the ARRAY statement earlier in this chapter for more information on using arrays.

Notes

1. **CMS, OS, VM/PC, and VSE**: The period preceding the decimal specification is not required with column output.

2. **AOS/VS, PRIMOS, and VMS:** You can align values and text within the field on the output line by including an alignment specification as part of the format:

-L left align
-C center
-R right align.

The field length is given by the width of the format. Include the alignment specification immediately after the format with no intervening spaces. For example, the following statement,

```
PUT DEPT $CHAR15.-C;
```

centers character values for DEPT within a field of fifteen spaces on the current output line. You can use format alignment specifications with character and numeric formats.

CMS, OS, VM/PC, and VSE: You cannot use an alignment specification.

3. **AOS/VS, PRIMOS, and VMS**: You can also enclose values for *n* in parentheses. This syntax can be useful if you want to use an expression or a SAS function, or to specify a negative integer value. For example,

```
PUT X +(-1) Y +(-1) Z;
```

eliminates the blank automatically inserted with list style output.

The following example uses expressions and SAS functions enclosed in parentheses in a PUT statement:

```
PUT @(N-FLOOR(X)) LASTNAME +(X-SUM(Y,Z,Q)) FRSTNAME;
```

This statement first uses the FLOOR function to take the largest integer value of X and then subtracts the result from the value of N. The pointer is moved to that position on the output line to write the value of LASTNAME. The pointer is then moved to a new position that is evaluated by subtracting the sum of three variables from the value of X; then the value of FRSTNAME is written.

4. **AOS/VS, PRIMOS, and VMS**: The n* format modifier is not available.

5. **AOS/VS, PRIMOS, and VMS**: SAS always takes the default action.

CMS, OS, VM/PC, and VSE: When overflow occurs, the action taken by SAS depends on which of the FILE statement options—FLOWOVER, DROPOVER, or STOPOVER—is in effect for the current output file. The default action is FLOWOVER. See the FILE statement for a description of these options.

6. **AOS/VS, PRIMOS, and VMS**: You can use the REPEAT function to repeat character constants. For example,

```
X=REPEAT('_', 131);
PUT X;
```

writes a line of 132 underscores. The n* format modifier is not available.

CMS, OS, VM/PC, and VSE: You can use the REPEAT function or the n* format modifier. For example,

```
PUT 132* '_';
```

or

```
X=REPEAT('_', 131);
PUT X;
```

With either method the PUT statement writes a line containing 132 underscores.

7. **CMS, OS, VM/PC, and VSE**: You can also specify an asterisk as the subscript to write the entire array, as the following program illustrates:

```
DATA _NULL_;
   INPUT FIRST $ SECOND $ THIRD $;
   ARRAY DAYS{3} $ FIRST SECOND THIRD;
   DO I=1 TO 3;
      IF DAYS{I}= 'TUES' THEN PUT DAYS{*};
   END;
   CARDS;
MON TUES WEDS
SAT SUN MON
TUES WED THURS
;
```

This program produces the following lines on the SAS log:

```
MON TUES WEDS
TUES WED THURS
```

RENAME Statement

Operating systems: All

You can use the RENAME statement in a DATA step to give variables new names in any data sets being created.

The form of the RENAME statement is

RENAME *oldname*=*newname*...;

where

oldname is the name of the individual variable you want to rename.

newname specifies the new name for the variable.

More than one set of names can appear. Since the new name takes effect in the output data set, use the old name in program statements in the current DATA step.

Here is an example:

```
DATA SUBSET;
   SET MASTER;
   RENAME OLD=NEW X1980=X1985;
   IF OLD>5;
   KEEP OLD X1980 Y Z;
PROC PRINT;
   VAR NEW X1985 Z;
```

In this example, the variable OLD, a variable in data set MASTER, is given the name NEW in data set SUBSET; variable X1980 from MASTER is given the name X1985. The existing names are used in the program statements that create SUBSET; the new names are used in the PROC PRINT step that prints SUBSET. See the RENAME= data set option in "SAS Files" for another way to change variable names.

RETAIN Statement

Operating systems: All

Introduction

The RETAIN statement causes a variable created by an INPUT or assignment statement to retain its value from the previous iteration of the DATA step. Without a RETAIN statement, the SAS System automatically sets variables created by INPUT or assignment statements to missing before each iteration of the DATA step. Naming variables read with a SET, MERGE, or UPDATE statement, or IN= variables in these statements, in a RETAIN statement has no effect.

You can use a RETAIN statement to specify initial values for variables. If a value appears in a RETAIN statement, variables appearing before it in the list are set to that value initially. (If you assign different initial values to the same variable by naming it more than once in a RETAIN statement, the last value is used.) You can also use a RETAIN statement to assign a sum variable an initial value other than the default value of 0.

The RETAIN statement can appear anywhere in the DATA step; its position has no effect. RETAIN is not an executable statement.

The form of the RETAIN statement is

RETAIN [*variables* ... [*initialvalue*]] ...**;**

where

variables names the variables whose values you want retained. If no variables are listed, for example,

```
RETAIN;
```

SAS retains the values of **all** variables in the DATA step (see **Cautions** below).

initialvalue optionally specifies an initial value for the preceding variables. On the first execution of the DATA step, variables preceding it (up to the preceding initial value) in the RETAIN statement are set to this value. If initial values are not specified, numeric variables appearing in a RETAIN statement are initially missing, and character variables are initially blank. If an initial value is specified, at least one variable name must precede it. RETAIN statements are useful for variables that compute totals and counts of other variables in the data set.

This is an example:

```
RETAIN MONTH1-MONTH5 1 YEAR 0 A B C 'ABC';
```

Variables MONTH1 through MONTH5 are set initially to a value of 1; YEAR starts out at 0; variables A, B, and C are each set to a value of 'ABC'.

Table 4.7 summarizes information on how the SAS System treats variable values with and without RETAIN statements.

Table 4.7 How the SAS System Retains Values

	Variables created in the DATA step	
	With INPUT or assignment statement	As sum variable
Without RETAIN statement	SAS sets all variables to missing before each iteration of the DATA step.	SAS sets the variable to zero before the first iteration of the Data step. Thereafter, the variable retains its current value until a new value becomes available (for example, in the next iteration of the sum statement).
With RETAIN statement	SAS sets variables to missing (or to the initial values given in the RETAIN statement) only before the first iteration of the DATA step. Thereafter, variables retain their values until new values become available (for example, through an assignment statement or the next iteration of the INPUT statement).	SAS sets the variable to zero or to the value given in the RETAIN statement before the first execution of the DATA step. Thereafter, the variable retains its value until a new value becomes available (for example, in the next iteration of the sum statement). Thus, the only purpose of naming a sum variable in a RETAIN statement is to give it an initial value other than zero.
	Variables read with SET, MERGE, or UPDATE statement	
Without RETAIN statement	WITH BY STATEMENT: SAS sets variables to missing before the first iteration of the SET, MERGE, or UPDATE statement for a given data set and before the first iteration of the statement for each BY group. Thereafter, the variables retain their values until new values become available (for example, through an assignment statement or through the next iteration of the SET, MERGE, or UPDATE statement). WITHOUT BY STATEMENT: SET: SAS sets variables to missing before the first iteration of the SET statement for a given data set. Thereafter, the variables retain their values until new values become available. MERGE: SAS sets variables to missing before the first iteration of the MERGE statement. Thereafter, the variables retain their values until new values become available. If a data set exhausts its observations, SAS sets variables contributed by that data set to missing.	
With RETAIN statement	Naming variables read with a SET, MERGE, or UPDATE statement in a RETAIN has no effect.	

See "DATA Step Applications" for more information about SET, MERGE, and UPDATE statements.

Example

In this example, the data set TIMECARD contains several observations for each SSN value. Different observations for a particular SSN value may have different values of the variable GRADE. You want to create a new data set, WORKERS, containing one observation for each SSN value. The observation must have the highest GRADE value of all observations for that SSN in TIMECARD.

```
PROC SORT DATA=TIMECARD;
   BY SSN;
DATA WORKERS;
   SET TIMECARD;
   BY SSN;
   IF FIRST.SSN THEN HIGHEST=.;
   RETAIN HIGHEST;
   HIGHEST=MAX(HIGHEST,GRADE);
   DROP GRADE;
   IF LAST.SSN THEN OUTPUT;
```

The variable HIGHEST is created in an assignment statement. Since the RETAIN statement tells SAS not to set the value of HIGHEST to missing each time a new observation is read, the statement

```
HIGHEST=MAX(HIGHEST,GRADE);
```

compares the previous value of HIGHEST to the value of GRADE in the current execution of the DATA step and assigns the higher value as the current value of HIGHEST. Since you want the highest GRADE value for each different SSN, not the highest value in the entire data set, the statement

```
IF FIRST.SSN THEN HIGHEST=.;
```

sets HIGHEST to a missing value for each new SSN value.

Output 4.22 shows the contents of data sets TIMECARD and WORKERS:

Output 4.22 Data Sets Before and After Highest Value Selected

DATA SET TIMECARD--MULTIPLE OBSERVATIONS PER SSN 1

OBS	SSN	GRADE
1	111442222	2
2	111442222	4
3	333115555	6
4	333115555	1

WORKERS--ONE OBSERVATION PER SSN AND HIGHEST VALUE OF GRADE 2

OBS	SSN	HIGHEST
1	111442222	4
2	333115555	6

Cautions

When just the keyword RETAIN appears without a variable list, all the variables created with INPUT or assignment statements are retained. Thus, data values can

be retained in observations that should have missing values. For example, consider this program:

```
DATA GROUP;
   INPUT X;
   RETAIN;
   IF X=1 THEN SEX='M';
   IF X=2 THEN SEX='F';
   CARDS;
data lines
;
```

When the value of X is either 1 or 2, all is well. But if X is 3, neither of the IF conditions is true, and SEX does not receive a new value. SEX retains its value from the preceding observation, producing an inaccuracy. Removing the RETAIN statement from this program results in SEX having a blank value when the value of X is other than 1 or 2.

RETURN Statement

Operating systems: All

The RETURN statement tells the SAS System to stop executing statements at the current point in the DATA step and to return to a predetermined point before continuing execution. The point to which the SAS System returns depends on the situation in which the RETURN statement appears.

- When a LINK statement has been executed, a RETURN statement causes the SAS System to return to the statement immediately following the LINK and continue executing.
- In a HEADER= group of statements, a RETURN statement causes the SAS System to return to the statement immediately following the last statement executed prior to beginning a new page and continue executing. (See the **FILE Statement** for a description of the HEADER= option.)
- Elsewhere in the DATA step, a RETURN statement causes the SAS System to return to the beginning of the step for another execution of the DATA step.

When a RETURN causes a return to the beginning of the DATA step, SAS first writes the current observation to any new data sets (unless OUTPUT statements are used in the step). SAS also increments the automatic variable _N_ by 1 and releases input lines held by a trailing @. Every DATA step has an implied RETURN as its last executable statement.

The form of the RETURN statement is

RETURN;

Here is an example of a RETURN statement used in an IF-THEN statement:

```
DATA SURVEY;
   INPUT X Y Z;
   IF X=Y THEN RETURN;
   X=Y+Z;
   A=X**2;
   CARDS;
data lines
;
```

When X equals Y, the RETURN statement is executed. The SAS System adds the observation to the data set and returns to the beginning of the DATA step. The two statements

```
X=Y+Z;
A=X**2;
```

are not executed.

When X is not equal to Y, the RETURN statement is not executed. The two assignment statements are executed before the observation is added to the data set.

The example below has a RETURN and an OUTPUT statement within an iterative DO group. The RETURN statement causes the SAS System to return to the beginning of the step when a particular condition is true, avoiding unnecessary iterations of the DO group.

```
DATA REPORT;
   INPUT A B C;
   DO X=1 TO 5;
      AX=A*X;
      IF AX>B THEN RETURN;
      OUTPUT;
      END;
   CARDS;
data lines
;
```

This DATA step produces multiple observations in SAS data set REPORT from each input line. Up to 5 observations can be generated from each observation. The statements in the DO group multiply the value of A by X and test the result (variable AX) to see whether the value is greater than the value of B. As long as the value of AX is less than or equal to B, SAS outputs that observation and continues execution of the DO group (up to 5 iterations). As soon as the value of AX becomes greater than B, the IF condition is true and the RETURN statement is executed. SAS returns to the beginning of the DATA step for a new observation, regardless of the number of iterations of the DO group that have occurred. Thus, unnecessary executions of the DO group are avoided since once the value of AX becomes larger than the value of B, it remains larger than B for all subsequent iterations of the loop for that observation.

See the **GO TO Statement** and the **LINK Statement** earlier in this chapter for other examples using the RETURN statement.

SELECT Statement

Operating systems: All

The SELECT statement in the DATA step allows the SAS System to execute one of several statements or groups of statements.

The SELECT statement begins a SELECT group; within the SELECT group, each WHEN statement identifies a SAS statement to be executed when a particular condition is true. At least one WHEN statement must be present. An optional OTHERWISE statement specifies a group of statements to be executed if no WHEN condition is met. An END statement ends a SELECT group.

The general form of the SELECT statement is

SELECT [(*selectexpression*)];
  WHEN (*whenexpression*) *statement*;
  ...
  [**OTHERWISE** *statement*;]
  END;

The following terms can appear in the SELECT statement:

selectexpression can be any valid SAS expression that evaluates to a single value.[1]

whenexpression can be any valid SAS expression. The way a WHEN expression is used depends on whether a SELECT expression is present.

If *selectexpression* is present, SAS first evaluates *selectexpression* and *whenexpression*. SAS then compares the two as though an equal sign connected them (performing a numeric-character or character-numeric conversion on the WHEN expression if necessary) and returns a value of true or false. If the comparison is true, *statement* (see below) is executed. When the comparison is false, execution proceeds to the next WHEN statement, if one is present, or the OTHERWISE statement, if one is present. If the result of all SELECT-WHEN comparisons is false and no OTHERWISE statement is present, SAS issues an error message.

If no SELECT expression is present, *whenexpression* is evaluated to produce a result of true (nonzero and nonmissing) or false (zero or missing). When the result is true, *statement* (see below) is executed. If more than one WHEN statement has a true WHEN expression, only the first WHEN statement is used. When the result is false, execution proceeds to the next WHEN statement, if one is present, or the OTHERWISE statement, if one is present. If the result of all WHEN expressions is false and no OTHERWISE statement is present, SAS issues an error message.

statement can be any executable SAS statement, including DO, SELECT, and null statements.

You can use a null statement in a WHEN statement to cause SAS to recognize a condition as true without

taking further action. For example, suppose a numeric variable A has a value of 1, 2, 3, or missing. This SELECT group leaves the value of X unchanged when A=2:

```
DATA NEW;
   SET OLD;
   X=UNIFORM(0);
   SELECT (A);
      WHEN (1) X=X*10;
      WHEN (2);
      WHEN (3) X=X*100;
      OTHERWISE X=1;
      END;
RUN;
```

You can use a null statement in an OTHERWISE statement, as in

```
OTHERWISE;
```

to prevent SAS from issuing an error message when all WHEN conditions are false.

The following example illustrates a SELECT statement with a SELECT expression:

```
*OPERATING SYSTEMS: ALL;
DATA PAYROLL;
   INPUT ID TYPE SALARY HRLYWAGE HRS;
   LABEL TYPE='10=SALARY, 20=HOURLY';
   SELECT (TYPE);
      WHEN (10) AMT=SALARY;
      WHEN (20) DO;
         AMT=HRLYWAGE*MIN(HRS,40);
         IF HRS>40 THEN LINK OVERTIME;
         END;
      OTHERWISE LINK ERROR;
      END;
   RETURN;
   OVERTIME: PUT 'CHECK TIMECARD ' _ALL_;
      RETURN;
   ERROR: AMT=.;
      _ERROR_=1;
      PUT 'PROBLEM OBSERVATION ' _ALL_;
      RETURN;
   CARDS;
68852 10 5000   .   .
46960 20 .    3.50 35
58342 20 .    4.00 55
64439 11 5200   .   .
;
```

In this example TYPE is a numeric variable with values of 10 and 20. When the value of TYPE is 10, the SAS System assigns the value of the variable SALARY to the variable AMT. When the value of TYPE is 20, SAS performs the calculations in the DO group. If the value of TYPE is anything else, SAS executes the LINK ERROR statement and performs the work indicated for an error in the data.

This example illustrates a SELECT statement without a SELECT expression. You can use it to do range checking, which must be done in a SELECT statement without a SELECT expression or in an IF statement.

```
*OPERATING SYSTEMS: CMS, OS, VM/PC, AND VSE;
DATA _NULL_;
   DO MON='JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
      'AUG', 'SEP';
      SELECT;
         WHEN (MON='JUN' | MON='JUL' | MON='AUG') PUT 'SUMMER ' MON=;
         WHEN (MON='MAR' | MON='APR' | MON='MAY') PUT 'SPRING ' MON=;
         OTHERWISE PUT 'FALL OR WINTER ' MON=;
         END; /* END OF SELECT GROUP*/
      END; /* END OF DO GROUP */
RUN;
```

Since there is no SELECT expression, SAS executes the first PUT statement for which the WHEN expression is true. If neither WHEN expression is true, SAS executes the OTHERWISE clause.

What happens if you use a SELECT expression in this example? SAS compares the result of the WHEN expression (1 or 0) to the value of MON in the current execution of the loop. Thus, for example, when MON has a value of JUN, the comparison is JUN='1'; the result of the comparison is false and the statement

```
PUT 'SUMMER ' MON=;
```

is not executed.

This example uses nested SELECT statements with complex expressions in WHEN statements. The outer SELECT group does not contain an OTHERWISE statement.

```
*OPERATING SYSTEMS: ALL;
DATA _NULL_;
   CENT='19';
   DO MONTH='JAN', 'FEB', 'MAR';
      DO DAY='MON', 'TUE', 'WED';
         DO YEAR='1984', '1985';
            SELECT (YEAR);
               WHEN (CENT||'84') DO;
                  SELECT (MONTH||DAY);
                     WHEN ('JANMON') PUT '1984 MON-JAN';
                     WHEN ('FEBTUE') PUT '1984 TUE-FEB';
                     WHEN ('MARWED') PUT '1984 WED-MAR';
                     OTHERWISE      PUT '1984 ' MONTH= DAY=;
                     END; /* MON||DAY */
                  END; /* CENT||84 */
               WHEN (CENT||'85') DO;
                  SELECT (MONTH||DAY);
                     WHEN ('JANWED') PUT '1985 WED-JAN';
                     WHEN ('FEBTUE') PUT '1985 TUE-FEB';
                     WHEN ('MARMON') PUT '1985 MON-MAR';
                     OTHERWISE      PUT '1985 ' MONTH= DAY=;
                     END; /* MONTH||DAY */
                  END; /* CENT||85 */
               END; /* SELECT YEAR */
            END; /* DO YEAR */
         END; /* DO DAY */
```

```
        END; /* DO MONTH */
RUN;
```

The WHEN expressions in the outer SELECT group are made up of the concatenated value of variable CENT and the characters '84' or '85'. You can also write the WHEN expressions as

```
WHEN ('1984') DO;
WHEN ('1985') DO;
```

The SELECT expressions in the inner SELECT groups are made up of the concatenated values of variables MONTH and DAY.

Note

1. **AOS/VS, VMS, and PRIMOS:** *Selectexpression* is required.
   CMS, OS, VM/PC, and VSE: *Selectexpression* is optional.

SET Statement

Operating systems: All

Introduction

The SET statement tells the SAS System to read observations from one or more SAS data sets. Use SET when you want to read, subset, concatenate, or interleave observations from existing SAS data sets into a new data set. The SET statement brings in all variables from the SAS data sets listed unless otherwise directed by a DROP= or KEEP= data set option. For example, the statements below create data set TWO as a copy of data set ONE:

```
DATA ONE;
   INPUT X Y Z;
   CARDS;
data lines
;
DATA TWO;
   SET ONE;
```

The SET statement is flexible and has a variety of uses in SAS programming. The examples in this description illustrate some common uses of the SET statement; for more advanced examples, including the use of more than one SET statement in a DATA step, see "DATA Step Applications."

The form of the SET statement is

SET [[*SASdataset*[**(***dsoptions* **IN=***name* **)**]...] [*setoptions*]]**;**

These terms and options can appear in the SET statement:

SASdataset
: names one or more existing SAS data sets from which to read observations each time the SET statement is executed. You can name up to 50 data sets in a SET statement. These are examples of valid SET statements:

```
SET FITNESS;
SET WORK.FITNESS;
SET SAVE.FITNESS;
SET HEALTH EXERCISE WELL;
SET MALES IN.FEMALES;
SET;
```

If you omit the data set name entirely in the SET statement, the statement is equivalent to:

```
SET _LAST_;
```

LAST is a special SAS data set name that refers to the most recently created SAS data set in your job.

See "SAS Files" for more information on using SAS data set names.

In general, SAS executes a SET statement by reading sequentially from the data sets listed. If more than one data set name is listed, SAS reads all observations from the first data set, then all from the second data set, and so on until all observations from all the data sets have been read. SAS thus treats all the data sets listed in the SET statement as one aggregate SAS data set. (See **Concatenating Data Sets** later in this section for more information.) You can cause SAS to process the observations from all data sets in sorted order by adding a BY statement (described in **Interleaving Data Sets** later in this section); you can cause SAS to process particular observations in the order you specify with the POINT= option, below.

(*dsoptions*)

specifies data set options in parentheses after each SAS data set name. For example,

```
DATA THREE;
   SET ONE(FIRSTOBS=100);
```

These options include those described in "SAS Data Sets" as well as the IN= data set option (described below), which is unique to the SET, MERGE, and UPDATE statements.

IN=*name*

creates a variable with the name given after the equal sign. Within the DATA step, the value of the variable is 1 if the data set contributed data to the current observation and 0 otherwise. The IN= data set option is most useful when the SET statement contains more than one data set name. By using the IN=option and a different variable name for each data set, you can tell which data set contributed the current observation. For example:

```
DATA FOUR;
   SET ONE(IN=INONE FIRSTOBS=100) TWO(IN=INTWO)
      THREE(IN=INTHREE);
   IF STATE=. THEN PUT STATE= INONE= INTWO= INTHREE=;
```

INONE has the value 1 when an observation is read in from ONE; otherwise, it has the value 0. The values of INTWO and INTHREE are determined similarly. Thus, you can see which data set contains the observation with a missing value for the variable STATE by examining the values of INONE, INTWO, and INTHREE.

IN= variables are not added to the data set being created.

setoptions

specifies the SET statement options described below.

POINT=*name*

creates a numeric variable whose value is the number of the observation in the input data set you want the current execution of the SET statement to process. You must specify the values of the POINT= variable (for example, by using the POINT= variable as the

index variable in a DO statement). The POINT= variable is available anywhere in the DATA step, but it is not added to any new SAS data set. This form of reading input data sets is called direct (or random) access by observation number. The data set read must be on disk.

For example, the statements below create a subset of ALL that contains only observations 3, 5, 7, and 4 in that order. The STOP statement tells SAS to stop building the data set after the DO loop is complete:

```
DATA SUBSET;
   DO N=3,5,7,4;
      SET ALL POINT=N;
      IF _ERROR_=1 THEN ABORT;
      OUTPUT;
      END;
   STOP;
```

When you use the POINT= option, you usually also include a STOP statement to stop processing the DATA step. See "DATA Step Applications" for an example where a STOP statement is not needed.

If you execute a random access SET statement with an invalid value of the POINT= variable, SAS sets the automatic variable _ERROR_ to 1. It is a good idea to check for this indication after a random access SET statement since it is more likely that your DATA step will go into an endless loop when this error goes undetected.

If more than one data set is listed in the SET statement, for example,

```
SET HOCKEY POLO POINT=N;
```

the POINT= variable retrieves observations from all the data sets as though they are concatenated. If the value of the POINT= variable is greater than the total number of observations, SAS sets the automatic variable _ERROR_ to 1.

The POINT= option **cannot** be used with a BY statement.

NOBS=*name*

creates a variable with the name given after the equal sign whose value is the total number of observations in the input data set. If more than one data set is listed in the SET statement, the value of the NOBS= variable is the total number of observations in the data sets listed. SAS assigns the value of the NOBS= variable automatically at compilation time. Thus, the NOBS= variable can appear before the SET statement. The variable is available in the DATA step but is not added to the new data set.[1]

See "DATA Step Applications" for an example using the NOBS= option.

END=*name*

creates a variable with the name given after the equal sign to contain an end-of-file indication. The variable, which is initialized to zero, is set to 1 when the SET statement reads the last observation of the input data set or concatenated data sets. This variable is not added to any new data set. If more than one data set is listed in the SET statement, the END= variable is set to 1 when the SET statement reads the last observation in the last data set listed.

This example uses the END= variable to write a message on the SAS log after the last observation of a concatenated series of SAS

data sets has been read and to create a macro variable whose value is the value of variable X in that observation:

```
DATA NEW;
   SET OLDA OLDB END=LAST;
   IF LAST THEN DO;
      PUT 'LAST OBSERVATION' X= Y=;
      CALL SYMPUT('MVARX',X);
      END;
RUN;
```

Copying Data Sets

To make a new copy called B of a SAS data set named A, write

```
DATA B;
   SET A;
```

This SET statement brings all variables in each observation of A into the DATA step to create SAS data set B.

Subsetting variables To make a copy containing a subset of variables, use DROP or KEEP specifications in any of three ways:

- Use a DROP= or KEEP= data set option in the SET statement:

```
DATA B;
   SET A(KEEP=X Y);
```

  In this case the subset is formed before the data are brought into the DATA step. This method saves computer resources, particularly when the original data set is large and you need only a few variables for the current data set.
- Use a DROP or KEEP statement in the DATA step:

```
DATA B;
   SET A;
   KEEP X Y;
```

  A DROP or KEEP statement controls which variables are output to SAS data sets; all variables in A are present in the DATA step and can be used in program statements, regardless of whether they are to be output to a data set. This method produces the same subset of variables in all output data sets.
- Use a DROP= or KEEP= option in the DATA statement:

```
DATA B(KEEP=X Y) C(KEEP=Y);
   SET A;
```

  This method allows you to output a different subset of variables to each data set specified. All variables in A are present in the DATA step and can be used in program statements.

Subsetting observations To make a copy containing a subset of observations, use program statements such as the subsetting IF, DELETE, or OUTPUT to control which observations are output to SAS data sets. For example, each of these DATA steps selects only the observations where the variable SEX has the value of M for the output data set:

```
DATA MALES;
```

```
   SET PEOPLE;
   IF SEX='M';

DATA MALES;
   SET PEOPLE;
   IF SEX='M' THEN OUTPUT;

DATA MALES;
   SET PEOPLE;
   IF SEX¬='M' THEN DELETE;
```

Creating new variables You can add program statements to create new variables as you copy a data set. For example, your original data set, MONTHLY, contains figures for several companies. The sales figures are represented by the variables MONTH1-MONTH12. You want to create a new variable to contain the total sales for the year for each company. You use a SET statement to tell SAS where to find the observations for the new data set and then create the new variable TOTAL with an assignment statement.

These SAS statements create a new data set YEARLY containing the same observations as MONTHLY plus a variable, TOTAL, which is the yearly sales figure.

```
DATA YEARLY;
   SET MONTHLY;
   TOTAL=SUM(OF MONTH1-MONTH12);
```

If you do not want the values of MONTH1-MONTH12 to be included in the YEARLY data set once you have created the total sales for the year, add a DROP statement to the DATA step. Data set YEARLY thus contains all the observations from data set MONTHLY, but excludes the monthly sales figures:

```
DATA YEARLY;
   SET MONTHLY;
   TOTAL=SUM(OF MONTH1-MONTH12);
   DROP MONTH1-MONTH12;
```

Concatenating Data Sets

Concatenation is processing two or more data sets one after the other as though they were a single large data set. To concatenate two or more SAS data sets to create a new one, name them in a SET statement. The new data set contains all the observations in all the input data sets listed.

Same variables In the simplest case, all the input data sets contain the same set of variables, which also are the variables in the new data set.

For example, say you have two data sets, each containing the same variables. One is made up of data for 1984; the other, data for 1985. You want to combine them into a single data set containing all observations for 1984 and 1985 with these statements:

```
DATA BOTHYEAR;
   SET Y1984 Y1985;
```

The number of observations in the new data set is the sum of the number of observations in the Y1984 data set and the number of observations in the Y1985 data set. The order of the observations in BOTHYEAR is all the observations of Y1984 followed by all the observations of Y1985. The new data set's variables are the same as those in the old data sets.

Different variables If the data sets in the SET statement contain different sets of variables, observations obtained from one data set have missing values for variables that are defined only in the other data set or data sets.

Different variable attributes Data sets listed in the SET statement may contain some of the same variables, but the variables may have different attributes. In that case SAS takes the action outlined in **Table 4.8**.

Table 4.8 Actions for Incompatible Variable Attributes

	INCOMPATIBLE ATTRIBUTE				
ACTION	TYPE	LENGTH	INFORMAT	FORMAT	LABEL
SAS issues an error message	X				
An explicitly specified attribute for the variable overrides a default attribute, regardless of the position of the data set in the list.		text below	X	X	X

For numeric variables, an explicitly specified length overrides a default length, regardless of the position of the data set in the list. For character variables, SAS uses the length in the first data set listed that contains the variable.

To change an attribute in the data set you are creating, specify the attribute before the SET statement, for example:

```
DATA A;
   NAME='MARY';
   *LENGTH OF VARIABLE NAME IS 4;
DATA B;
   NAME='SUZANNE';
   *LENGTH OF VARIABLE NAME IS 7;
DATA C;
   LENGTH NAME $ 10;
   SET A B;
   *LENGTH OF VARIABLE NAME IS 10;
```

See the LENGTH statement earlier in this chapter for more examples of changing variable lengths.

Interleaving Data Sets

Interleaving is an operation that combines data sets in sorted order. To interleave two or more data sets, use a BY statement as well as a SET statement. The data sets listed in the SET statement must be sorted by the variables used in the BY statement. SAS reads all the observations in a given BY group from the first data set listed, then all the observations in that BY group from the second data set listed, and so on. The new data set contains all the observations in the input data sets in sorted order by the variables in the BY statement. SAS treats data sets with different variables and variables with different attributes in different data sets the same way that it does in a concatenation operation.

Suppose you have two data sets, one for each department in your company. You want to combine the two input data sets, interleaving observations from each

one, to create a new data set sorted by the variable LASTNAME. The number of observations in the new data set is the sum of the number of observations in each of the original data sets. Each data set must be sorted by the variable LASTNAME. These statements perform the interleaving operation:

```
PROC SORT DATA=DEPT1;
   BY LASTNAME;
PROC SORT DATA=DEPT2;
   BY LASTNAME;
DATA ALLEMPL;
   SET DEPT1 DEPT2;
   BY LASTNAME;
```

Note: although the term "merge" is often used to mean "interleave," SAS reserves merge for a matching operation described under the MERGE statement.

Note

1. **AOS/VS, PRIMOS, and VMS**: You can use the NOBS= option regardless of whether you have used the POINT= option.

   CMS, OS, VM/PC, and VSE: You must use the POINT= option in order to use the NOBS= option.

STOP Statement

Operating systems: All

You can use the STOP statement to stop processing a SAS DATA step. The observation being processed when the STOP statement is encountered is not added to the SAS data set.

The form of a STOP statement is

STOP;

The STOP statement does not affect the execution of any subsequent DATA or PROC steps. Execution continues from the first DATA or PROC statement found after the STOP statement. (In batch mode, use the ABORT statement if you want the SAS System to stop processing the entire SAS job.)

Here is an example:

```
DATA COUNT1;
   INPUT X Y Z;
   IF _N_=250 THEN STOP;
   CARDS;
more than 250 data lines
PROC PRINT;
```

The value of the automatic variable _N_ corresponds to the number of the current execution of the DATA step; thus, in this example the value of _N_ corresponds to the number of the observation being processed. During the 250th execution of the DATA step, the STOP statement is executed and SAS stops building the data set. SAS then executes the PROC PRINT step. Since the 250th observation was being processed when the STOP statement was encountered and was not added to the data set, data set COUNT1 contains 249 observations.

Sum Statement

Operating systems: All

Sum statements add the result of an expression to an accumulator variable as observations are being added to the data set.

The form of a sum statement is

variable + *expression*;

where

variable
: specifies the name of the accumulator variable. The variable must be a numeric variable; any valid SAS variable name can be used. The variable is automatically set to 0 before the first observation is read. Its value is retained from one execution to the next, just as if it had appeared in a RETAIN statement. To initialize a sum variable to a value other than 0, include it in a RETAIN statement with an initial value. (See the RETAIN statement description for more information.)

expression
: is any valid SAS expression. The expression is evaluated and the result added to the accumulator variable. When the evaluation of the expression results in a missing value, it is treated as zero.

The sum statement is equivalent to using the SUM function and a RETAIN statement as shown below:

```
variable=SUM(variable,expression,0);
RETAIN variable 0;
```

Here is an example:

```
DATA ACCUM;
   INPUT X Y Z;
   IF X=4 THEN N+1;
   IF N=250 THEN STOP;
   CARDS;
data lines
;
```

Each time the sum statement

```
N+1;
```

is executed, 1 is added to the value of N. When 250 observations with an X value of 4 have been read, the value of N is 250. The statement

```
IF N=250 THEN STOP;
```

tells SAS to stop building the data set when N equals 250.

Some more examples of sum statements are given below:

```
BALANCE+(-DEBIT);
```

This sum statement subtracts the DEBIT amount from BALANCE. The plus (+) is required in a sum statement; the statement

```
BALANCE-DEBIT;
```

is incorrect.

In the sum statement

```
SUMXSQ+X*X;
```

the result of X*X is added to SUMXSQ each time the sum statement is executed. The following statements:

```
NX+X¬=.;
NX+(X¬=.);
```

show the use of the comparison operator. When the value of X is not missing, the expression is true (has the value 1). When X is missing, the expression's value is 0. Thus, NX contains the number of observations processed with a nonmissing X value. See "SAS Expressions" for more information about logical expressions.

UPDATE Statement

Operating systems: All

Introduction

The UPDATE statement combines observations from two SAS data sets in a manner similar to the MERGE statement, but UPDATE performs the special function of updating a master file by applying transactions. The UPDATE statement must be accompanied by a BY statement giving the name of an identifying variable by which to match observations in the two data sets. The data set containing the master file and the data set containing the transactions must both be sorted by this identifying variable or variables, and the master data set must not contain observations with duplicate values of the identifying variable. The new data set contains one observation for each observation in the master data set; if any transaction observations were not matched with master observations, these become new observations in the new data set as well.

The form of the UPDATE statement is

UPDATE *masterdataset* [(*dsoptions***IN=***variable1*)] *transactiondataset* [(*dsoptions* **IN=***variable2*)][**END=***variable*];

where

masterdataset
: names the SAS data set used as the master file. Only one observation in this data set can have each value of the variable(s) in the BY statement.

transactiondataset
: names the SAS data set containing transactions to be applied to the master. This data set must contain the BY variable(s), but multiple observations with the same BY value can occur.

dsoptions
: give SAS more information about the data set to which the data set options refer. See "SAS Files" for the data set options that are available.

IN=*variable*
: creates a variable with the name given after the equal sign. A different IN= variable should be associated with each data set. The UPDATE operation indicates which data sets contributed data to the current observation by setting values of this variable to 1 or 0. Specify IN= in parentheses after the names of the data sets in the UPDATE statement as shown below:

```
DATA THREE;
   UPDATE ONE(IN=INONE) TWO(IN=INTWO);
   BY ID;
```

INONE has the value 1 when information in the current observation comes from the master data set ONE; otherwise, it has

the value 0. The value of INTWO is determined similarly. Both variables INONE and INTWO are equal to one if both data sets contribute information.

END=*name*
is used optionally to create a variable with the name given after the equal sign to let you know when the last observation in the UPDATE operation is being processed. The variable, which is initialized to zero, is set to 1 when the UPDATE statement is on the last observation. This variable is not added to any new data set.

Example

You have a SAS data set containing your master file and another SAS data set containing updates—your transaction file. There may be several transactions for each master observation. Usually, both data sets contain the same variables, but the transaction data set need not contain variables that will not be updated, and it may contain variables to be added to the master data set. If a variable is being updated in some, but not all of the observations, its value is missing in observations where it is not to be changed. Both data sets are sorted by the identifying variable. Below is an example:

Output 4.23 Master and Transaction Data Sets

DATA OLDMASTR 1

OBS	ID	NAME	SEX	AGE	WEIGHT
1	01	JONES	M	46	116.8
2	04	MILLER	F	59	132.2
3	30	PARKER	M	29	111.3
4	49	SMITH	M	34	209.1
5	87	WILSON	F	30	98.3

DATA TRANS 2

OBS	ID	AGE	WEIGHT	NAME
1	01	47	.	
2	30	.	108.4	
3	49	35	215.1	
4	87	.	.	CAMERON
5	87	.	104.1	

Since both data sets are sorted in order of ID, these statements perform the update:

```
DATA NEW;
   UPDATE OLDMASTR TRANS;
   BY ID;
```

Here is the resulting data set NEW:

Output 4.24 New Data Set

DATA NEW 3

OBS	ID	NAME	SEX	AGE	WEIGHT
1	01	JONES	M	47	116.8
2	04	MILLER	F	59	132.2
3	30	PARKER	M	29	108.4
4	49	SMITH	M	35	215.1
5	87	CAMERON	F	30	104.1

Note that

- since the variable SEX is not updated, it does not appear in the TRANS data set
- variables need not appear in the same order in the two data sets
- only values to be updated are specified in the transaction data set; missing values in TRANS do not change the original values in OLDMASTR
- multiple transaction records (as with ID 87) are all applied to the master before the record is output to NEW
- if a record is not to be updated, it need not appear in the transaction data set (see ID 04).

Updating Values to Missing

To set the value of any variable in the master data set to missing, you must use the special missing value, underscore (_), as the variable's value in the transaction data set. (See the MISSING statement description or "Missing Values" for more information.)

For example, suppose data set TRANS, described in the previous example, were created with these statements:

```
DATA TRANS;
   MISSING _;
   INPUT ID $ AGE WEIGHT NAME $;
   CARDS;
```

To set the value of AGE in JONES's master record to a missing value, use a special missing value underscore in the transaction data set:

Output 4.25 Transaction Data Set with Special Missing Value

DATA TRANS WITH SPECIAL MISSING VALUE 4

OBS	ID	AGE	WEIGHT	NAME
1	01	_	30.0	
2	49	35	215.1	
3	87	.	.	CAMERON
4	87	.	104.1	

When you update MASTER with TRANS, JONES's age is updated to a missing value:

```
DATA NEW;
```

```
UPDATE OLDMASTR TRANS;
BY ID;
```

Output 4.26 New Data Set with Special Missing Value

NEW WITH SPECIAL MISSING VALUE 5

OBS	ID	NAME	SEX	AGE	WEIGHT
1	01	JONES	M	.	30.0
2	04	MILLER	F	59	132.2
3	30	PARKER	M	29	111.3
4	49	SMITH	M	35	215.1
5	87	CAMERON	F	30	104.1

When an underscore is coded as the value of a character variable in the transaction data set, it is considered a special missing value and updates the value in the master data set to missing. No MISSING statement is needed in the DATA step that creates the transaction data set.

Details and additional examples of using UPDATE are given in "DATA Step Applications."

Chapter 5

SAS® Expressions

INTRODUCTION

An *expression* is a sequence of operators and operands forming a set of instructions that are performed to produce a result value. The operands are variable names and constants. The operators are special-character operators, functions, and grouping parentheses. (There are 113 SAS functions that can perform a variety of roles and are especially useful as programming shortcuts. See "SAS Functions" for a complete discussion.)

Expressions can be *simple* (using only one operator) or *compound* (using more than one operator). The following are examples of expressions:

```
X+1
3
LOG(Y)
PART/ALL*100
1-EXP(N/(N-1))
AGE<100
STATE='NC' | STATE='SC'
A=B=C
```

Use expressions in DATA step programming statements for transforming variables, creating new variables, conditional processing, calculating new values, assigning new values, and bit testing.

SAS CONSTANTS

A *SAS constant* is a number, or a character string in single quotes (or double quotes if the DQUOTE system option is in effect), or other special notation that indicates a fixed value. Constants are also called *literals*. In the assignment statement

```
X=7;
```

the number 7 is a numeric constant, and X is a variable name.

SAS uses five kinds of constants:

- numeric
- character
- date, time, and datetime numeric
- hexadecimal character
- hexadecimal numeric.

The hexadecimal constants are special constants discussed separately at the end of this chapter in **Special Constants and Operators**.

Constants can be used in assignment, sum, IF, SELECT, RETAIN, PUT, and ERROR statements, and as values for certain procedure options.

Numeric Constants

A numeric constant is simply a number that appears in a SAS statement. Most numeric constants are written just as numeric data values are. The numeric constant in the expression

```
PART/ALL*100
```

is 100.

Numeric constants can use a decimal point, a minus sign, and E-notation (scientific notation). For example,

```
1
1.23
01
-5
1.2E23
0.5E-10
```

In E-notation, the number before the E is scaled to the power of ten indicated by the number after the E; for example, 2E4 is the same as $2x10^4$ or 20000.[1]

A constant representing an ordinary numeric missing value is written as a single decimal point (.). A constant representing a special numeric missing value is written as two characters: a decimal point followed by a letter (for example, .B) or an underscore (._).

Character Constants

A character constant consists of 1 to 200 characters enclosed in single quotes (or double quotes if the DQUOTE system option is in effect). For example, in the statement

```
IF NAME='TOM' THEN DO;
```

'TOM' is a character constant.

If a character constant includes a single quote, write it in a SAS expression as two consecutive single quotes; SAS treats it as one. For example, if you want to specify the character value TOM'S as a constant, you enter:

```
NAME='TOM''S'
```

If DQUOTE is in effect, you can write

```
NAME="TOM'S"
```

A constant representing a missing character value consists of a blank enclosed in quotes (for example, ' ').

Date, Time, and Datetime Constants

To express a date, time, or datetime value as a constant, use the same notation used in the informats and formats: TIME., DATE., and DATETIME. (See "SAS Informats and Formats.") Enclose the formatted value in single quotes, and follow it with a D (date), T (time), or DT (datetime). Here are some examples:

```
'1JAN1980'D
'01JAN80'D
'9:25'T
'9:25:19'T
'18JAN80:9:27:05'DT
```

A date constant might be used in an expression like this:

```
IF BEGIN='01JAN1981'D THEN END='31DEC1981'D;
```

SAS OPERATORS

SAS operators are symbols that request a comparison, logical operation, or arithmetic calculation. SAS uses two major kinds of operators: prefix operators and infix operators.

A *prefix operator* is an operator that is applied to the variable, constant, function, or parenthesized expression immediately following it, for example, -6. The plus sign (+) and minus sign (-) can be used as prefix operators. The word NOT or the symbols ¬ or ^ are also prefix operators (see **Logical Operators**). Some examples of prefix operators used with variables, constants, functions, and parenthesized expressions are shown below:

```
+Y
-25
-COS(ANGLE1)
+(X*Y)
```

Infix operators apply to an operand on each side of an operator, for example, 6<8. There are four general kinds of infix operators: arithmetic; comparison; logical or Boolean; and others (minimum, maximum, and concatenation). Bit-mask testing is a special case of a comparison operation and is described in **Bit Testing**.

Number of Operators: Simple and Compound Expressions

When there is only one operator in an expression (for example, A/B), it is a *simple expression*. When there is more than one operator in an expression (for example, 1−EXP(N/N−1)), it is a *compound expression*.

Order of Performing Operations

The rules describing the order of evaluation for compound expressions are

Rule 1 Expressions within parentheses are evaluated before those outside. 18/3*2 is the same as 6*2 or 12, but 18/(3*2) is equivalent to 18/6 or 3. −(X**2) is equivalent to the mathematical expression $-(X^2)$, but (−X)**2 is the same as the mathematical expression $(-X)^2$.

Rule 2 Higher priority operations are performed first. The table below shows the priority groups:

Group I

**
+ prefix only
− prefix only
¬ or ^ (NOT)²
>< (MINIMUM)
<> (MAXIMUM)

Group II

*
/

Group III

+ infix only
− infix only

Group IV

| |

Group V

< <= = ¬= >= > ¬> ¬<

Group VI

& (AND)

Group VII

| (OR)

Note that plus (+) and minus (−) can be either prefix or infix operators. A plus or a minus is a prefix operator only when it appears at the beginning of an expression or when it is immediately preceded by a left parenthesis or another operator.

Rule 3 For operators with the same priority, the left operation is done first. There are two exceptions to this:

1. For the highest priority group (Group I), the right operation is done first.
2. When two comparison operators surround a quantity, the expression is evaluated as if an AND is present. For example, the expression:

```
12<AGE<20
```

is evaluated as if it is written:

```
12<AGE & AGE<20
```

Arithmetic Operators

Arithmetic operators indicate that an arithmetic calculation is performed. The arithmetic operators are

** raise to a power
* multiplication
/ division
+ addition
− subtraction

For example, A**3 means raise the value of A to the power 3. The expression 2*Y means multiply 2 by the value of Y.

Note: the asterisk (*) is always necessary to indicate that multiplication is performed; thus, 2Y is **not** a valid expression.

If a missing value is an operand for an arithmetic operator, the result is a missing value.

Comparison Operators

Comparison operators propose a relationship between two quantities and ask SAS to determine whether or not that relationship holds. If it does hold (in other words, if it is true), the result of carrying out the operation is the value 1; if it does not hold (in other words, if it is false), the result is the value 0. The comparison operators are[3]

= or EQ	equal to
¬= or NE	not equal to
> or GT	greater than
< or LT	less than
>= or GE	greater than or equal to
<= or LE	less than or equal to

Consider the expression A<=B. If A has the value 4 and B has the value 3, then A<=B has the value 0 (false). If A is 5 and B is 9, then the expression has the value 1 (true). If A and B each have the value 47, then again the relationship holds and the expression assumes the value 1.

Comparison operators appear frequently in IF statements. For example,

```
IF X<Y THEN C=5;
   ELSE C=12;
```

Comparisons are also used in expressions in assignment statements. For example, the above statements could be recoded:

```
C=5*(X<Y)+12*(X>=Y);
```

Since quantities inside parentheses are evaluated before any operations are performed on them, the expressions (X<Y) and (X>=Y) are evaluated first, and the result (1 or 0) is substituted for the parenthesized expression. Therefore, if X=6 and Y=8:

```
C=5*(1)+12*(0)
```

```
C=5
```

Note: in a comparison operation a missing value of any kind compares smaller than any other numeric value.

Character comparisons Comparisons are performed on character-valued as well as numeric operands, although the comparison always yields a numeric result (1 or 0). Character operands are compared character-by-character from left to right. Character order is determined by machine-collating sequence (see a note at the end of "The SORT Procedure" for both ASCII and EBCDIC collating sequences).

The expression 'GRAY'>'ADAMS' is true, as is the expression 'JONES, C.'<'JONES, CLYDE'.

Two character values of unequal length are compared as if blanks are attached to the end of the shorter value before the comparison is made. So, 'FOX' is equivalent to 'FOX '. (However, 'FOX' is not equivalent to ' FOX' because blanks at the beginning and in the middle of a character value are meaningful to SAS.)

For example, say you want to create a data set that includes a variable called NAME. You only want to include observations for which the NAME value begins with the letters S through Z. The statement

```
IF NAME>='S';
```

is equivalent to

```
IF NAME>='S
```

where blanks are extended after the S to the length of the variable NAME. Thus, the data set being created contains all observations with NAME values beginning with S through Z, that is, all values alphabetically greater than 'S '.

To compare only the first letter of the NAME values to the letter S, you can use a colon (:) after the comparison operator. SAS then truncates the longer value to the length of the shorter value for the comparison. For example, the SAS statement

```
IF NAME>:'S';
```

compares the first character of NAME values with S. Any observations having NAME values beginning with T or later in the alphabet are included in the data set being created.

Note that SAS truncates and extends values only during the comparison. The values themselves keep their lengths. Any of the eight comparison operators listed above can be used with the colon in this way.

Logical Operators

Logical operators, also called *Boolean operators*, are usually used in expressions to link sequences of comparisons. The logical operators are

&	AND
|	OR
¬	NOT
^	NOT[2]

If **both** of the quantities surrounding an AND are 1 (true), then the result of the AND operation produces a 1; otherwise, the result is 0. For example, the expression

```
A<B & C>0
```

is true (has the value 1) only when **both** A<B is 1 (true) **and** C>0 is 1 (true); that is, when A is less than B **and** C is positive.

If **either** of the quantities surrounding an OR is 1 (true), then the result of the OR operation is 1 (true); otherwise, the OR operation produces a 0. For example, the expression

```
A<B | C>0
```

is true (has the value 1) when A<B is 1 (true) regardless of the value of C. It is also true when the value of C>0 is 1 (true), regardless of the values of A and B. It is true, then, when either or both of those relationships hold.

Be careful when using the OR operator with IF statements:

```
IF X=1 OR 2;
```

is not the same as:

```
IF X=1 OR X=2;
```

The first IF statement is always true. X=1 is evaluated first and the result may be 1 or 0; however, the 2 is evaluated as 2=2, and since 2=2 is always true, the whole expression is true. The second IF statement is not necessarily true.

The prefix operator NOT is also a logical operator. The result of putting NOT in front of a quantity whose value is 0 (false) is 1 (true). That is, the result of negating a false statement is 1 (true). For example, if X=Y is 0 (false) then NOT(X=Y) is 1 (true). The result of NOT in front of a quantity whose value is missing is also 1 (true). The result of NOT in front of a quantity with a nonzero, nonmissing value is 0 (false); in other words, the result of negating a true statement is 0 (false).

For example,

```
NOT(NAME='SMITH')
```

is equivalent to

```
NAME NE 'SMITH' .
```

By DeMorgan's law, the NOT of an AND is the OR of the NOTs. The NOT of an OR is the AND of the NOTs. That is, ¬(A&B) is equivalent to ¬A|¬B, and ¬(A|B) is equivalent to ¬A&¬B. For example,

```
NOT(A=B & C>D)
```

is equivalent to

```
A NE B | C LE D
```

Do not use the keyword NOT as a variable name.

Other Operators

The operators in this category are >< (MIN), <> (MAX), and || (concatenation).

The >< and <> operators are surrounded by two quantities. The result is the quantity that is the minimum if >< is the operator or the maximum if <> is the operator. For example, if A<B, then A><B has the value of A.

The || operator concatenates two character values. For example, if the variable COLOR has a value of BLACK, and the variable NAME has a value of JACK, then:

```
LENGTH GAME $ 9;
GAME=COLOR || NAME;
```

results in GAME having a value of BLACKJACK.

If ALPHA='TEK' and MODEL='4662' then the statement

```
DEVICE=ALPHA || MODEL;
```

results in DEVICE='TEK4662' .

You can concatenate several character variables in one expression. If A='ONE' and B='AND' and C='ONLY', then

```
D=A || B || C;
```

results in D='ONEANDONLY'.

You can also concatenate character constants. For example, if OLDNUM='123', then

```
NEWNUM=OLDNUM || '80';
```

causes NEWNUM to be assigned the value of '12380'.

The concatenation operator does not trim leading or trailing blanks. The expression

```
NAME='JOHN   '||'SMITH'
```

produces the value

```
'JOHN   SMITH'.
```

Use the TRIM function (described in "SAS Functions") if you want SAS to trim trailing blanks from values before concatenating them.

SPECIAL CONSTANTS AND OPERATORS

Hexadecimal Constants

You can also represent hexadecimal character and numeric values as constants.

Hexadecimal character constants A character hex constant is a string of an even number of hex characters enclosed in single quotes, followed immediately by an X. For example,

```
'E2C1E2'X
'534153'X
```

A comma can be used to make the string more readable, but it is not part of, and does not alter, the hex value. If the string contains a comma, the comma must separate an even number of hex characters within the string. For example,

```
IF VALUE='F1F2,F3F4'X THEN DO;
IF VALUE='3132,3334'X THEN DO;
```

Hexadecimal numeric constants Hexadecimal numeric constants can be specified in SAS statements. A numeric hex constant starts with a numeric digit (usually a zero), can be followed by more hexadecimal digits, and ends with the letter X. If the constant does not begin with a numeric digit, SAS can treat it as a variable name. The constant can contain no more than 16 valid hexadecimal digits (0-9, A-F). Here are some examples of numeric hex constants:

```
0C1X 0B37X 322X 0C4X 9X
```

Numeric hex constants can be used in a DATA step like this:

```
DATA;
   INPUT ABEND PIB2.;
   IF ABEND=0C1X OR ABEND=0B0AX THEN DO;
```

Bit Testing

Bit testing is a special comparison operation that tests internal bits in a value's representation. The general form of the operation is

expression=*bitmask*

You use a bit mask to test bits. The bit mask is a string of 0s, 1s, and periods in quotes, immediately followed by a B. 0s test whether the bit is off; 1s test for the bit on; and periods ignore the bit. Commas and blanks can be inserted in the bit mask for readability without affecting its meaning.

Both character and numeric variables can be the subject of bit testing. When testing a character value, SAS aligns the leftmost bit of the mask with the leftmost bit of the string; the test proceeds through the corresponding bits, moving to the right. When SAS tests a numeric value, the value is truncated to an integer and converted to a fixed-point number (2^{31}-1). The rightmost bit of the mask is aligned with the rightmost bit of the number, and the test proceeds through the corresponding bits, moving to the left.

Here is an example of a test of a character variable:

```
IF A='..1.0000'B THEN DO;
```

If the third bit of A (counting from the left) is on, and the fifth through eighth bits are off, then the comparison is TRUE and the expression results in 1. Otherwise the comparison is FALSE and the expression results in 0. Here is another example:

```
DATA;
   INPUT @88 BITS $CHAR1.;
   IF BITS='10000000'B THEN CATEGORY='A';
   ELSE IF BITS='01000000'B THEN CATEGORY='B';
   ELSE IF BITS='00100000'B THEN CATEGORY='C';
```

Note that bit masks are a special convention used with the equal sign (=) comparison operator. They **cannot** be used as bit literals in expressions. For example, the statement

```
X='0101'B;
```

is not valid.

SPECIAL TOPICS

Numeric-Character Conversion

SAS automatically converts character variables to numeric variables, and numeric variables to character variables, according to these rules:

- If you use a character variable with an operator that requires numeric operands (for example, the plus sign), SAS converts the character variable to numeric.
- If you use a comparison operator to compare a character variable and a numeric variable, the character variable is converted to numeric.
- If you use a numeric variable with an operator that requires a character value (for example, the concatenation operator), the numeric value is converted to character using the BEST12. format.
- If you use a numeric variable on the left side of an assignment statement and a character variable on the right, the character variable is converted

to numeric. In the opposite situation, where the character variable is on the left and the numeric is on the right, SAS converts the numeric variable to character using the BEST*n*. format, where *n* is the length of the variable on the left.

Whenever SAS performs an automatic conversion, it prints a message on the SAS log warning that the conversion took place.

If converting a character variable to numeric produces invalid numeric values, SAS assigns a missing value to the result, prints an error message on the log, and sets the value of the automatic variable _ERROR_ to 1.

Working with Fractional Values and Variables

A common error with calculations involves fractional values. SAS uses the floating point instructions provided by the hardware for all numeric variable calculations. The machine stores floating point numbers only to a fixed precision. Fractional values that cannot be represented exactly in binary or hexadecimal are not stored exactly.

Even if only a fractional difference exists, the values of fractions may not be identical, and the value stored varies according to the hardware used. You can check the hexadecimal representation of numbers on your machine by displaying them with the SAS format HEX.

Because SAS is used for diverse applications, there is no best way to round or "fuzz" values that would satisfy all users. In cases where fractional values are critical, use the INT or ROUND function to tell SAS the precision you need.

You should also be careful when working with calculated fractional variables. For example, the MEANS, SUMMARY, TABULATE, and UNIVARIATE procedures with the FREQ statement always use integers, and if not supplied with an integer value, the FREQ statement truncates the value supplied.

Suppose you have a variable that was calculated using the formula

```
X=0.3*100;
```

X's value is not exactly 30, as you would expect, but it is close enough for most applications. However, if used as a frequency variable, only the integer portion 29 is used. In this case, round the frequency variable before using it. If you want 29 used with values such as 29.6 or 29.556, then use a smaller unit for rounding.

These precautions do not affect most applications. When working with integer values, values can be stored exactly and no precautions are needed.

NOTES

1. Numeric constants larger than (10**32)-1 must be specified in scientific notation.
2. **AOS/VS, PRIMOS, and VMS**: use the ^ operator.
   CMS, OS, VM/PC, and VSE: use the ¬ operator.
3. **CMS, OS, VM/PC, and VSE**: The following operators are also available:

¬> or NG	not greater than
¬< or NL	not less than.

Chapter 6

SAS® Functions

INTRODUCTION

A *SAS function* is a routine that returns a value computed from one or more *arguments*. Each SAS function has a keyword name. To invoke a function, write its name and then the argument or arguments for which the calculation is to be performed enclosed in parentheses:

```
FUNCTIONNAME(argument,argument)
```

Here are some examples of functions and arguments:

- The INT function yields the integer value of the variable CASH:

```
INT(CASH)
```

- The SUM function adds the values of the variables CASH and CREDIT:

```
SUM(CASH,CREDIT)
```

- In this example, the result of one function (SUM) is an argument of another function (MIN):

```
MIN(SUM(CASH,CREDIT),1000)
```

  The MIN function compares the value of the first argument to that of the second argument and returns the smaller of the two values. The first argument is the result of the SUM function, which adds the variables CASH and CREDIT. The second argument is a constant, 1000.

SAS functions are used in DATA step programming statements and in some statistical procedures as expressions or parts of expressions. Consider the preceding examples; they must be entered as parts of SAS statements in order to have any effect. The INT function might be used as follows:

```
DATA INCOME;
   INPUT STORE DATE DATE7. CASH 8.2 CREDIT 8.2;
   DOLLARS = INT(CASH);
   CARDS;
data lines
;
```

A new variable, DOLLARS, is assigned the value that is the result of the INT function.

You can use a SAS function anywhere you would use a SAS expression, not just in assignment statements. The next example uses the SUM function in an IF/THEN statement:

```
DATA _NULL_;
   SET INCOME;
   IF SUM(CASH,CREDIT) > 1000 THEN PUT
      STORE ' IS AN OUTSTANDING ACME COMPANY STORE';
```

Remember as you read the function descriptions in this chapter that SAS functions must always be used in SAS programming statements.

SAS functions are especially useful as programming shortcuts for many numeric calculations and manipulations of character and numeric data. They are also useful for creating new variables from existing data. For example, say that you want to create a variable called LEAST whose value is the smaller of

- the sum of ten variables (X1, X2, ..., X10)
- a variable called Y.

You could create the variable LEAST by writing:

```
TOTX=X1+X2+X3+X4+X5+X6+X7+X8+X9+X10;
IF TOTX<Y THEN LEAST=TOTX;
ELSE LEAST=Y;
```

It is faster, however, to write:

```
LEAST=MIN(SUM(OF X1-X10),Y);
```

SAS functions fall into a number of categories: arithmetic, truncation, mathematical, trigonometric and hyperbolic, probability, sample statistics, random number, character, date and time, state and ZIP code, system, and special. **Table 6.1**, later in this chapter, lists the functions by category.

FUNCTION ARGUMENTS

Function arguments can be simply variable names

```
MAX(CASH,CREDIT)
```

or constants

```
SQRT(1500)
REPEAT('----+',16)
```

or they can be expressions, including expressions involving other functions:

```
MIN((ENROLL-DROP),(ENROLL-FAIL))
```

Some functions require no argument, some only one argument, and some operate on several arguments. (No function allows more than 2000 arguments.) All expression arguments are evaluated before a function is called. For example, 2*LOG(X+Y) is twice the natural logarithm of the sum of the value of X and the value of Y. The addition is performed first, then the logarithm of the sum is calculated, and that result is multiplied by 2.

Normally, when there is more than one argument, they must be separated by commas. However, if the arguments of a function are all variables and are in a sequence (for example, X1 through X5, or X, Y, and Z), the function can also be written in one of these two forms:

```
FUNCTIONNAME(OF variable1-variablen)
FUNCTIONNAME(OF variable variable variable ...)
```

For example, any of these forms is correct:

```
SUM(OF X1-X100 Y1-Y100)
SUM(OF X  Y  Z)
SUM(X1,X2,X3,X4)
```

You **cannot** use these forms to list sequential variable arguments of a function:

```
FUNCTIONNAME(variable1 variable2 ...)
FUNCTIONNAME(variable1variable2...)
FUNCTIONNAME(variable1-variablen)
```

Each of these statements produces an error message:

```
Y=SUM(X1 X2 X3);
Y=SUM(XYZ);
```

Some functions require that their arguments have values only in a certain range; for example, the argument of the LOG function must be greater than zero. Most functions (with the exception of those producing sample statistics) do not permit missing values as arguments. If the value of a function's argument is inadmissible (that is, missing or outside a certain range), SAS prints an error message and sets the result to a missing value. If an argument of a function that produces sample statistics is missing, that value is not included in the calculation of the statistic.

In general, the allowed range of the arguments is machine-dependent (as is true, for example, with the EXP function). Note also that for some probability functions, combinations of extreme values in the arguments can cause convergence problems. When this occurs, a missing value is returned and an error message is issued.

FUNCTION RESULTS

The resulting or *target* variable for a function is usually character if the arguments are character and numeric if the arguments are numeric. The PUT function is an exception as its result is a character value regardless of the type of argument.

By default, the length of the target variable for most functions is eight for numeric target variables and 200 for character target variables. For example,

```
LEAST=(MIN(SUM OF X1-X10),Y);
```

LEAST, the target variable, is a numeric variable of length 8. Functions to which the default target variable lengths do not apply are shown below:

Function	Target Variable Type	Target Variable Length
DIF	numeric	length of first argument
INPUT	character	width of informat
	numeric	8
LAG	numeric	length of first argument
PUT	character	width of format
SUBSTR	character	length of first argument
TRIM	character	length of argument

FUNCTION CATEGORIES

Table 6.1 Functions by Category

Arithmetic functions	
ABS	returns absolute value
DIM	returns the number of elements in an array
MAX	returns the largest value
MIN	returns the smallest value
MOD	calculates the remainder
SIGN	returns the sign of the argument or 0
SQRT	calculates the square root
Truncation functions	
CEIL	returns the smallest integer>=argument
FLOOR	returns the largest integer<=argument
FUZZ	returns the integer if the argument is within 1E−12
INT	returns the integer value (truncates)
ROUND	rounds a value to the nearest roundoff unit
Mathematical functions	
DIGAMMA	computes the derivative of the log of the GAMMA function
ERF	is the error function
ERFC	returns the complement of the ERF function
EXP	raises e (~2.71828) to a specified power
GAMMA	produces the complete gamma function
LGAMMA	calculates the natural logarithm of the GAMMA function of a value
LOG	produces the natural logarithm (base e)
LOG2	calculates the logarithm to the base 2
LOG10	produces the common logarithm
Trigonometric and hyperbolic functions	
ARCOS	calculates the arc cosine
ARSIN	calculates the arc sine
ATAN	calculates the arc tangent
COS	calculates the cosine
COSH	calculates the hyperbolic cosine
SIN	calculates the sine
SINH	calculates the hyperbolic sine
TAN	calculates the tangent
TANH	calculates the hyperbolic tangent

Probability functions

BETAINV	inverse beta distribution function
GAMINV	inverse gamma distribution function
POISSON	Poisson probability distribution function
PROBBETA	beta probability distribution function
PROBBNML	binomial probability distribution function
PROBCHI	chi-squared probability distribution function
PROBF	*F* distribution function
PROBGAM	gamma probability distribution function
PROBHYPR	hypergeometric probability distribution function
PROBIT	inverse normal distribution function
PROBNEGB	negative binomial probability distribution function
PROBNORM	standard normal probability distribution function
PROBT	Student's *t* distribution function

Sample statistic functions

CSS	calculates the corrected sum of squares
CV	calculates the coefficient of variation
KURTOSIS	gives the kurtosis
MAX	returns the largest value
MIN	returns the smallest value
MEAN	computes the arithmetic mean (average)
N	returns the number of nonmissing arguments
NMISS	returns the number of missing values
RANGE	returns the range
SKEWNESS	gives the skewness
STD	calculates the standard deviation
STDERR	calculates the standard error of the mean
SUM	calculates the sum of the arguments
USS	calculates the uncorrected sum of squares
VAR	calculates the variance

Random number functions

NORMAL	generates a normally distributed pseudo-random variate
RANBIN	generates an observation from a binomial distribution
RANCAU	generates a Cauchy deviate
RANEXP	generates an exponential deviate
RANGAM	generates an observation from a gamma distribution
RANNOR	generates a normal deviate
RANPOI	generates an observation from a Poisson distribution
RANTBL	generates deviates from a tabled probability mass function

RANTRI	generates an observation from a triangular distribution
RANUNI	generates a uniform deviate
UNIFORM	generates a pseudo-random variate uniformly distributed on the interval (0,1)

Character functions

COLLATE	generates a string of characters in collating sequence
COMPRESS	removes characters from a character variable argument
INDEX	searches for a pattern of characters
INDEXC	finds the first occurrence of any one of a set of characters
LEFT	left-aligns a character string
LENGTH	returns the length of a character argument
REPEAT	repeats characters
REVERSE	reverses characters
RIGHT	right-aligns a character string
SCAN	scans for words
SUBSTR	extracts a substring
TRANSLATE	changes characters
TRIM	removes trailing blanks
UPCASE	converts to uppercase
VERIFY	validates a character value

Date & time functions

DATE	returns today's date as a SAS date value
DATEJUL	converts a Julian date to a SAS date value
DATEPART	extracts the date part of a SAS datetime value or literal
DATETIME	returns the current date and time of day
DAY	returns the day of the month from a SAS date value
DHMS	returns a SAS datetime value from date, hour, minute, and second
HMS	returns a SAS time value from hour, minute, and second
HOUR	returns the hour from a SAS datetime or time value or literal
INTCK	returns the number of time intervals
INTNX	advances a date, time, or datetime value by a given interval
JULDATE	returns the Julian date from a SAS date value or literal
MDY	returns a SAS date value from month, day, and year

MINUTE	returns the minute from a SAS time or datetime value or literal
MONTH	returns the month from a SAS date value or literal
QTR	returns the quarter from a SAS date value or literal
SECOND	returns the second from a SAS time or datetime value or literal
TIME	returns the current time of day
TIMEPART	extracts the time part of a SAS datetime value or literal
TODAY	returns the current date as a SAS date value
WEEKDAY	returns the day of the week from a SAS date value or literal
YEAR	returns the year from a SAS date value
YYQ	returns a SAS date value from the year and quarter

State & ZIP code functions

FIPNAME	converts FIPS code to state name (all uppercase)
FIPNAMEL	converts FIPS code to state name in upper- and lowercase
FIPSTATE	converts FIPS state codes to two-character postal code
STFIPS	converts state postal codes to FIPS state codes
STNAME	converts state postal codes to state names (all uppercase)
STNAMEL	converts state postal codes to state names (upper- and lowercase)
ZIPFIPS	converts ZIP codes to FIPS state codes
ZIPNAME	converts ZIP codes to state names (all uppercase)
ZIPNAMEL	converts ZIP codes to state names (upper- and lowercase)
ZIPSTATE	converts ZIP codes to two-letter state codes

System functions

CMS	invokes a CMS command and returns the status code
SYSPARM	returns the system parameter string
TSO	invokes a TSO command and returns the status code

Special functions

DIF	calculates the first difference for the *n*th lag
INPUT	defines an informat for a character value
LAG	lags variable values
PUT	specifies an output format for a value
SASVER	returns the version of SAS
SYMGET	returns the value of a macro variable

Notes on Sample Statistic Functions

There are fifteen functions whose results are sample statistics of the values of the arguments. The functions correspond to the statistics produced by the MEANS procedure and the computing method for each statistic is discussed in "SAS Descriptive Procedures." In each case, the statistic is calculated for the nonmissing values of the arguments.

Notes on Random Number Functions

You can generate random numbers for various distributions using the random number functions. The random number functions use an argument to select an initial seed value, which initializes the random number stream. Depending on the value of the argument, one of three types of initialization is used:

Argument	Type of Initialization
0	On the first execution of a function, giving 0 for an argument initializes the stream with a seed equal to a computer clock observation and returns an observation generated with this seed. On subsequent executions the function returns an observation generated with the current seed.
>0	On the first execution of the function, the argument is used as the current seed to initialize the stream and return an observation. On subsequent executions, an observation generated with the current seed is returned.
<0	When negative values are given for the argument, a new clock observation is used as the current seed to generate an observation at **every** execution of the function.

Note: the performance of the generators has not been evaluated when the argument is either zero or negative, and negative seed values may produce correlated random number sequences.

Although the current seed changes each time the function is executed, the value of the argument remains unchanged. You cannot control seed values, and therefore, random numbers, after the initialization. If you want more control of the number streams, use the CALL subroutine corresponding to the random number function (see **CALL Subroutines**) rather than the function itself.

CALL subroutines SAS provides a series of subroutines that give you more control over the seed stream and the random number stream than is possible with the random-number-generating functions. There is a subroutine corresponding to every random number function except NORMAL and UNIFORM.

The random-number-generating subroutines are invoked with CALL statements. The general form of a CALL statement is

CALL *subroutine*(*seed*,*variate*);

where *subroutine* is the name of any SAS random-number-generating function (except NORMAL or UNIFORM), *seed* is the name of a variable to hold the current seed values, and *variate* is the name of a variable to hold the generated variates. The *seed* variable should be initialized prior to the first execution of the CALL statement, for example, in an assignment or RETAIN statement.

After an execution of the CALL statement, *seed* contains the current seed in the stream (that is, the seed that will generate the next number), and *variate* contains the generated number.

Using the CALL subroutines rather than the random number functions allows you to initialize more than one random number stream in a DATA step. By comparison, more than one set of random numbers can be created with the random number functions, but they all come from one stream (only one stream is initialized). The example below illustrates this.

```
DATA A;
   RETAIN SEED1 SEED2 1613218064;
   DO I=1 TO 5;
      X1=RANUNI(SEED1);
      X2=RANUNI(SEED2);
      OUTPUT;
      END;
PROC PRINT;
   TITLE 'USING A RANDOM NUMBER FUNCTION';
```

Output 6.1 Random Number Functions Can Initialize Only One Stream

USING A RANDOM NUMBER FUNCTION 1

OBS	SEED1	SEED2	I	X1	X2
1	1613218064	1613218064	1	0.800831	0.770936
2	1613218064	1613218064	2	0.009603	0.498510
3	1613218064	1613218064	3	0.442188	0.646033
4	1613218064	1613218064	4	0.500457	0.731599
5	1613218064	1613218064	5	0.558058	0.500674

Notice that although the initial values of SEED1 and SEED2 are the same (1613218064), the numbers generated and held in X1 and X2 are different. This is because only one stream was initialized; the program ignores the value given for SEED2, and uses the current seed in the stream begun by the first RANUNI statement instead. Thus, the first value of X2 is not the result of a seed value of 1613218064; it is the result of an unknown seed.

Below is an example that shows that more than one stream can be initialized if a CALL subroutine is used rather than a function.

```
DATA;
   RETAIN SEED4 SEED5 1613218064;
   DO I=1 TO 5;
      CALL RANUNI(SEED4,X4);
      CALL RANUNI(SEED5,X5);
      OUTPUT;
      END;
PROC PRINT;
   TITLE 'USING A CALL SUBROUTINE INSTEAD OF A FUNCTION';
```

Output 6.2 A CALL Subroutine Can Initialize More Than One Stream

USING A CALL SUBROUTINE INSTEAD OF A FUNCTION 1

OBS	SEED4	SEED5	I	X4	X5
1	1719772190	1719772190	1	0.800831	0.800831
2	1655573359	1655573359	2	0.770936	0.770936
3	20623105	20623105	3	0.009603	0.009603
4	1070543076	1070543076	4	0.498510	0.498510
5	949591638	949591638	5	0.442188	0.442188

After the DATA step is completed, the values of X4 and X5 are identical for each observation because the CALL statement initialized a second seed stream that began with the same value as the first stream. By initializing SEED4 and SEED5 with different seeds, you can obtain observations from independent streams. Notice that with the CALL subroutines it is possible to see what the current seed is at any time; this is not possible with the random number functions.

FUNCTION DESCRIPTIONS

ABS: returns the absolute value

```
ABS(argument)
```

The *argument* must be a numeric value. This function returns a positive number of the same magnitude as the original value of the argument.

Examples:

```
DATA;
   X=ABS(2.4);
   PUT X;
RUN;
```

The value returned is 2.4.

```
X=ABS(-3);
```

The value returned is 3.

ARCOS: calculates the arccosine

```
ARCOS(argument)
```

The *argument* for this function must be a numeric value between −1 and +1. The function returns the arccosine (inverse cosine) of the value of the argument. The result is in radians.

For the examples

```
X=ARCOS(1);
X=ARCOS(0);
X=ARCOS(-.5);
```

the values returned are 0, 1.570796, and 2.094395, respectively.

ARSIN: calculates the arcsine

```
ARSIN(argument)
```

The *argument* for this function must be a numeric value between −1 and +1. The function returns the arcsine (inverse sine) of the value of the argument. The result is in radians.

For the examples

```
X=ARSIN(0);
X=ARSIN(1);
X=ARSIN(-.5);
```

the following values are returned: 0, 1.570796, and −0.523599, respectively.

ATAN: calculates the arctangent

```
ATAN(argument)
```

The *argument* for this function must be a numeric value. The result is the arctangent (inverse tangent) of the value of the argument. The result is in radians.

Examples:

```
X=ATAN(1);
X=ATAN(0);
X=ATAN(-9);
```

The values returned for these examples are 0.7853982, 0, and −1.46014, respectively.

BETAINV: computes the inverse of the cumulative beta distribution

```
BETAINV(p,a,b)
```

where

$$0 \leq p \leq 1,\ a > 0, \text{ and } b > 0 \quad .$$

The function BETAINV returns the *p*th quantile from a beta distribution with density:

$$\Gamma(a+b)x^{a-1}(1-x)^{b-1}/\Gamma(a)\Gamma(b) \quad .$$

For example,

```
BETAINV(.001,2,4);
```

returns a value of .0101. The beta distribution is related to many common statistical distributions including the *F* distribution (Abramowitz and Stegun 1964).

CEIL: returns the smallest integer≥argument

```
CEIL(argument)
```

The *argument* must be a numeric value. This function returns the smallest integer that is greater than or equal to the value of the argument. If the argument's value is within 10^{-12} of an integer, the function results in that integer. Examples:

```
DATA;
   X=CEIL(2.1);
   PUT X;
RUN;
```

The value returned is 3.

```
X=CEIL(3);
```

The value returned is 3.

```
X=CEIL(-2.4);
```

The value returned is −2.

```
X=CEIL(-1.6);
```

The value returned is −1.

CMS: invokes a CMS command and returns the status code

```
CMS(command)
```

The CMS function invokes a single CMS or CP command. The *command* is a character string enclosed in quotes corresponding to a CMS or CP command. CP commands must be preceded by CP. The function returns the status code set after the command executes. The status code is always 0 for CP commands. Any CMS subset command may be invoked with the CMS function.

Example:

```
RC=CMS('FILEDEF DAILY DISK DAY DAILY A');
```

The variable RC contains the status code returned after the CMS FILEDEF command specified by the CMS function is executed.

Operating systems: CMS and VM/PC

COLLATE: generates a collating sequence character string

```
COLLATE(n,m,l)
```

COLLATE returns a string of characters, beginning with the *n*th character, from either the EBCDIC or ASCII collating sequence.[1] The COLLATE function returns a maximum of 200 characters. If the string requested is longer, COLLATE returns only the first 200 characters from the collating sequence.

Define the length of the string by specifying either *m*, the last character in the sequence you want, or *l*, the number of characters you want. For example, the statements

```
X=COLLATE(240,249);
X=COLLATE(240,,10);
```

both give X the value 0123456789.

If neither *m* nor *l* is specified, the default length is 200.[2] If both *m* and *l* are specified, *l* is ignored.

COMPRESS: removes specific characters from a character expression

```
COMPRESS(argument)
COMPRESS(argument1,argument2)
```

When only one argument is specified, COMPRESS returns it after removing all the blanks. When two arguments are specified, the COMPRESS function returns

the first argument after removing from it all characters specified in the second argument.

For example, the following statements eliminate the blanks from the value of A:

```
A='AB C D ';
B=COMPRESS(A);
```

B's value is ABCD.

The statements below remove the special characters .;() from the value of X:

```
X='A.B (C=D);';
Y=COMPRESS(X,'.;()');
```

Y's value is AB C=D.

COS: calculates the cosine

```
COS(argument)
```

The *argument* for this function must be a numeric value. The result is the cosine of the value of the argument. The value of the argument is assumed to be in radians.

The examples

```
X=COS(.5);
X=COS(0);
X=COS(3.14159/3);
```

return 0.8775826, 1, and .5, respectively.

COSH: calculates the hyperbolic cosine

```
COSH(argument)
```

The *argument* for this function must be a numeric value. The result is the hyperbolic cosine of the value of the argument. The result is equivalent to

```
(EXP(argument)+EXP(-argument))/2
```

.

For the examples

```
X=COSH(0);
X=COSH(-5);
X=COSH(.5);
```

the values returned are 1, 74.20995, and 1.27626, respectively.

CSS: calculates the corrected sum of squares

```
CSS(argument,argument, ...)
```

The *arguments* for this function must be numeric values. For two or more nonmissing arguments, the function calculates the corrected sum of squares of the nonmissing arguments.

For the examples

```
X=CSS(4,2,3.5,6);
```

and

```
X=CSS(4,2,3.5,6,.);
```

the value returned in either case is 8.1875.

You can use an abbreviated argument list preceded by OF:

```
X=CSS(OF Y1-Y30);
```

The variable names Y1 to Y30 represent numeric values used to calculate the result.

See "SAS Descriptive Procedures" for a definition of this statistic.

CV: calculates the coefficient of variation

```
CV(argument,argument, ...)
```

The *arguments* for this function must be numeric values. The function calculates the coefficient of variation for the two or more nonmissing arguments that you give.

For example,

```
X=CV(5,8,9,3,6);
```

returns the value 38.50754. You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

DATE: returns today's date as a SAS date value

```
DATE()
```

The DATE function, which may also be written:

```
TODAY()
```

produces today's date as a SAS date value.

For example, suppose that you work in a company's billing office and are responsible for preparing a list of customers whose bills are more than 15 days overdue. You maintain a SAS data set containing customer information, including the variable DATEDUE. DATEDUE contains the date a payment was due as a SAS date value. The following DATA step prepares the list of overdue accounts.

```
DATA _NULL_;
   SET CUSTOMER;
   TDAY=DATE();
   IF (TDAY-DATEDUE)> 15 THEN DO:
      PUT 'AS OF' TDAY DATE7. 'ACCOUNT #' ACCOUNT
      'IS MORE THAN 15 DAYS OVERDUE.';
```

Note that the TDAY variable is output in DATE7. format. If a format is not assigned to TDAY, the value printed will be a SAS date value (the number of days since January 1, 1960).

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DATEJUL: converts a Julian date to a SAS date value

```
DATEJUL(Juliandate)
```

The DATEJUL function converts a Julian date to a SAS date value. A Julian date has the form *yyddd,* where *yy* is the year 19*yy* and *ddd* is the day of the year. The DATEJUL function is useful any time you want to represent values stored as Julian dates in some other form. For example, the SAS data set SUBSCRIB contains information on a newspaper's customers. Beginning subscription dates are stored in the variable BEGIN in Julian form. You want to print a report for each newspaper route showing the date each customer began receiving the paper in WORDDATE. format. First, you must convert the BEGIN value to a SAS date value.

```
DATA _NULL_;
   SET SUBSCRIB;
   START=DATJUL(BEGIN);
   PUT @1 CUSTOMER @30 START WORDDATE20.;
```

If the last three digits of the argument's value represent a value less than 1 or greater than 366, the function prints an "invalid argument" message.

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DATEPART: extracts the date part of a SAS datetime value

```
DATEPART(datetime)
```

The DATEPART function extracts the date part of a SAS *datetime* value or datetime literal. The date returned is a SAS date value. For positive dates, the statement

```
DATE=DATEPART(datetime);
```

is equivalent to

```
DATE=INT(datetime/(24*60*60));
```

For example, say that a computing center's accounting office issues statements that list charges for each day. A billing program keeps track of connect time used and stores the information as SAS datetime values. The variable CONN shows when a user's computer session began. To create a SAS data set of charges by date, use the DATEPART function.

```
DATA DAILY;
   SET ACCOUNTS;
   KEEP USER CHARGES SERVDATE;
   SERVDATE=DATEPART(CONN);
```

The value of the new variable SERVDATE is a SAS date value. To print SERVDATE in readable form, you must use a SAS date format.

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DATETIME: returns the current date and time of day

```
DATETIME()
```

The DATETIME function produces the current time of day and date as a SAS datetime value. Suppose you are writing a program that will execute whenever

a bank customer uses an automatic teller machine. The program must record the transaction for the bank's data base, and show the date and time value of the transaction, as well as the deposit or withdrawal information. Your program might include these statements:

```
DATA _NULL_;
   WHEN=DATETIME();
   more SAS statements
   PUT @1 ID @30 WHEN DATETIME17.;
```

Note that the value of WHEN is formatted with the DATETIME. format in order to print in readable form. If no format is used, the value printed is a SAS datetime value (the number of seconds since midnight, January 1, 1960).

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DAY: returns the day of the month from a SAS date value

```
DAY(date)
```

The DAY function produces the day of the month from a SAS date value or date literal. Suppose you need a DATA step that updates a large SAS data set on odd-numbered days. The program is run every day, and you want it to print a notice saying "NO UPDATE TODAY" on even-numbered days. The DATA step might look like this:

```
DATA INVNTRY;
   TDAY=DATE();
   DOM=DAY(TDAY);
   DAYNUM=MOD(DOM,2);
   IF DAYNUM>0 THEN DO;
   UPDATE INVNTRY NEWDATA;
   more SAS statements
   ELSE PUT TDAY DATE7. 'NO UPDATE TODAY';
```

First, the DATE function is used to assign the current date to the variable TDAY. Then the day of the month is extracted from TDAY by the DAY function and assigned to the target variable DOM. Next, the MOD function is used to find the remainder of DOM divided by 2. On odd-numbered days, the remainder is greater than 0, so the DO group updating the master data set executes. On even-numbered days, the remainder is 0, so the message is printed instead.

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DHMS: returns a SAS datetime value from date, hour, minute, and second

```
DHMS(date,hour,minute,second)
```

The DHMS function produces a SAS datetime value from *date, hour, minute,* and *second* values. The date argument can be either a variable or a SAS date literal. For example, a utility company samples electricity usage at 100 residences for six months. On selected days, data are recorded every second. The company wants to analyze the data by date, hour, minute, and second across all house-

holds, so they need a data set that can be sorted by a datetime variable. The DHMS function helps them create the data set.

```
DATA USAGE;
   INFILE READINGS;
   INPUT DATE DATE7. HOUR 2. MINUTE 2. SECOND 2. KW 10.1 HOUSE 4.;
   DTID= DHMS(DATE,HOUR,MINUTE,SECOND);
   KEEP DTID KW HOUSE;
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

DIF: calculates the first difference for the n^{th} lag

```
DIFn(argument)
```

A family of DIF functions, named DIF1, DIF2, ..., DIF100 is available for numerical arguments only. (DIF1 can also be written DIF.) The result of the DIF*n* function is the current value of *argument* minus the current LAG*n* value of *argument*; that is:

```
DIFn(X)=X-LAGn(X).
```

Consider this DATA step:

```
DATA TWO;
   INPUT X @@;
   Z=LAG(X);
   D=DIF(X);
   CARDS;
1 2 6 4 7
;
```

Data set TWO looks like this:

```
X  Z  D
1  .  .
2  1  1
6  2  4
4  6 -2
7  4  3
```

Note: the function DIF2(X) is not equivalent to the second difference DIF(DIF(X)).

DIFs are only generated when a DIF function executes; they are not automatically generated for each observation. If the DIF function executes conditionally in an IF-THEN statement, only values from those observations meeting the IF condition are differenced.

If the argument is an array name, each array element is differenced separately.

DIGAMMA: computes the digamma function

```
DIGAMMA(x)
```

The argument must be a positive integer. The DIGAMMA function computes the derivative of the log of the gamma function, that is, DIGAMMA(x)=$\Gamma'(x)/\Gamma(x)$. The value of this function is undefined for nonpositive integers.

Example:

```
Y=DIGAMMA(1);
```

returns

```
Y= -.5772
```

which is the negative of Euler's constant.

DIM: returns the number of elements in an array

```
DIMn(arrayname)
```

where *n* is the dimension for which you want to know the number of elements.

The DIM function returns the number of elements in a one-dimensional array or the number of elements in a specified dimension of a multidimensional array.[3]

If the array is one-dimensional, or if you want the first dimension of a multidimensional array, no *n* value need be specified. For example, suppose you define this multidimensional array:

```
ARRAY MULT{5,10,2} MULT1-MULT100;
```

The value of DIM(MULT) is 5, while DIM2(MULT) is 10, and DIM3(MULT) is 2.

The DIM function is convenient because you can use it to avoid having to change the upper bound of the iterative DO group each time you change the number of elements in the array. Here is an example for a one-dimensional array. The statements

```
DATA NEW;
   INPUT WEIGHT SEX HEIGHT STATE CITY;
   ARRAY BIG {5} WEIGHT SEX HEIGHT STATE CITY;
   DO I=1 TO DIM(BIG);
      PUT 'THE VALUE OF BIG{I} FOR I=' I 'IS ' BIG{I};
      END;
   CARDS;
83 2 64 14 6230
;
```

produce this output:

```
THE VALUE OF BIG{I} FOR I=1 IS 83
THE VALUE OF BIG{I} FOR I=2 IS 2
THE VALUE OF BIG{I} FOR I=3 IS 64
THE VALUE OF BIG{I} FOR I=4 IS 14
THE VALUE OF BIG{I} FOR I=5 IS 6230
```

ERF: error function

```
ERF(argument)
```

The result of this function is $2/\sqrt{(\pi)}$ times the definite integral from 0 to *argument* of e^{-x^2}.

The ERF function can be used to find the probability (P) that a normally distributed random variable with mean 0 and standard deviation 1 will take on a value less than X

```
P=.5+ERF(X/SQRT(2))/2    .
```

This is equivalent to PROBNORM(X).

Examples:

```
DATA;
   Y=ERF(1);
   PUT Y;
RUN;
```

returns the value .842701.

```
Y=ERF(-1);
```

returns the value −.842701.

ERFC: returns the complement of the ERF function

```
ERFC(argument)
```

The function ERFC represents the complement to the ERF function (that is, 1−ERF).

EXP: raises e (2.71828) to a specified power

```
EXP(argument)
```

The value of the argument is limited by the machine that you use. For example, using IBM the value of the argument must be less than 174.673, approximately. This function raises e to the power specified by the argument. e, the base of natural logarithms, is approximately 2.71828.

Examples:

```
Y=EXP(1);
```

returns the value 2.71828.

```
Y=EXP(0);
```

returns the value 1.

FIPNAME: converts FIPS code to state name (uppercase)

```
FIPNAME(FIPS)
```

The FIPNAME function takes a numeric FIPS code and returns the corresponding 20-character state name in uppercase. For example,

```
X=FIPNAME(37);
PUT X=;
```

produces the line

```
X=NORTH CAROLINA
```

FIPNAMEL: converts FIPS code to state name in upper- and lowercase

```
FIPNAMEL(FIPS)
```

The FIPNAMEL function takes a numeric FIPS code and returns a twenty-character state name in upper- and lowercase. For example,

```
X=FIPNAMEL(37);
```

```
PUT X=;
```

produces the line

```
X=North Carolina
```

FIPSTATE: converts FIPS code to two-character postal code

```
FIPSTATE(FIPS)
```

The FIPSTATE function takes the numeric FIPS state code and returns the two-character postal code. For example,

```
X=FIPSTATE(37);
PUT X=;
```

produces the line

```
X=NC
```

FLOOR: returns the largest integer≤argument

```
FLOOR(argument)
```

This function results in the largest integer less than or equal to the value of the argument. If the argument's value is within 10^{-12} of an integer, the function results in that integer.

Examples:

```
DATA;
   X=FLOOR(2.1);
   PUT X;
RUN;
```

returns the value 2.

```
X=FLOOR(-2.4);
```

returns the value −3.

```
X=FLOOR(3);
```

returns the value 3.

```
X=FLOOR(-1.6);
```

returns the value −2.

FUZZ: returns the integer if the argument is within 1E−12

```
FUZZ(argument)
```

The FUZZ function returns an integer value if the *argument* is within 1E−12 of the integer; that is, if the difference between the integer and argument is less than 1E−12. For example, these statements:

```
DATA _NULL_;
   X=5.9999999999999;
   Y=FUZZ(X);
   PUT Y= 15.13;
```

produce this line:

```
Y=6.0000000000000
```

GAMINV: inverse of the PROBGAM function

```
GAMINV(p,eta)
```

where

$0 \leq p < 1$, and $\eta > 0$.

The GAMINV function computes a value of *X* such that

$$P = \int_0^x t^{\eta-1} e^{-t} dt / \Gamma(\eta) \quad .$$

GAMMA: produces the complete gamma function

```
GAMMA(x)
```

The result of this function is the definite integral:

$$\int_0^{oo} t^{x-1} e^{-t} d^t \quad .$$

The value of *x* must be a positive number less than 57.5744, approximately. If it is an integer, then GAMMA(*x*) is *(x-1)!*. This function is commonly denoted by $\Gamma(x)$.

Example:

```
DATA;
   X=GAMMA(6);
   PUT X;
RUN;
```

produces a value of 120.

HMS: returns a SAS time value from hour, minute, and second

```
HMS(hour,minute,second)
```

The HMS function produces a SAS time value from *hour, minute,* and *second* values. Suppose that the utility company in the example for the DHMS function (above) wanted to determine what time of day is, on the average, the time of peak electrical usage. The HMS function helps them set up the results so they can find the average.

```
DATA USAGE;
   INFILE READINGS;
   INPUT DATE DATE7. HOUR 2. MINUTE 2. SECOND 2. KW 10.1 HOUSE 4.;
   HRID= HMS(HOUR,MINUTE,SECOND);
   KEEP HRID DATE KW HOUSE;
   more SAS statements
```

The result is printed as a SAS time value (the number of seconds since 12:00 midnight).

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

HOUR: returns the hour from a SAS datetime or time value or literal

```
HOUR(time)
HOUR(datetime)
```

The HOUR function operates on a SAS *time* or *datetime* value, or a SAS time or datetime literal, to produce a numeric value containing the hour. For example, suppose employees in your department often run a large SAS job that prints between 200 and 300 pages of output. The information is valuable, but the printer is tied up for nearly an hour every time someone runs the job. The HOUR function allows you to display a message any time someone begins to run the job between 7 a.m and 5 p.m.

```
DATA _NULL_;
   NOW=TIME();
   H=HOUR(NOW);
   IF 7≤H≤17 THEN DO;
   PUT 'PLEASE WAIT UNTIL AFTER 5 P.M. TO RUN THIS JOB.';
```

See **Using SAS Date, Time, and Datetime Informats and Formats** for a discussion of date and time values, informats, formats, and literals.

INDEX: searches the first argument for the character string specified by the second argument

```
INDEX(argument1,argument2)
```

The INDEX function searches *argument1*, from left to right, for the first occurrence of the string specified in the second argument, and returns the position in argument1 of the string's first character. If the string is not found in argument1, INDEX returns a value of 0. For example,

```
A='ABC.DEF (X=Y)';
B='X=Y';
X=INDEX(A,B);
```

The value of X is 10, the starting position in A of the string specified by B, X=Y.

INDEXC: finds the first occurrence in the first argument of any character present in any of the remaining arguments

```
INDEXC(argument1,argument2,...argumentn)
```

The INDEXC function searches the first argument, from left to right, for the first occurrence of any character present in any of the other arguments. INDEXC returns the position in argument1 of the character found. INDEXC returns a value of 0 if none of the characters in argument 2 through n are found in argument1.

For example, the following statements find the first numeric or special character in the character string assigned to the variable, A.

```
A='ABC.DEP (X2=Y1)';
X=INDEXC(A,'0123456789',';()=. ');
```

The value of X is 4 because the period (.), the fourth character in A, is the first character from the remaining arguments that INDEXC finds in A.

INPUT: defines an informat for a value

```
INPUT(argument,informat)
```

This function allows you to "read" *argument* using any informat specified by the second argument. The informat specified determines whether the result is numeric or character. For example,

```
RELEASE='76.6';
NRELEASE=INPUT(RELEASE,4.1);
```

results in NRELEASE having the numeric value 76.6.

The variable or constant specified for *argument* can be character or numeric. By specifying a character informat with a numeric value, it is possible to convert numeric values to character; however, we do not recommend that you do this. If you use the INPUT function to convert numeric values to character, an implicit conversion takes place using the BEST12. informat. The result of the conversion is right-aligned and is padded with blanks if necessary. The conversion may result in a value you are not expecting. If you need to convert numeric values to character values, we recommend that you use the PUT function.

INT: returns the integer value (truncates)

```
INT(argument)
```

This function truncates the decimal portion of the value of the argument. The integer portion of the value of the argument remains. If the argument's value is within 10^{-12} of an integer, the function results in that integer. If the value of *argument* is positive, INT(*argument*) has the same result as FLOOR(*argument*). If the value of *argument* is negative, INT(*argument*) has the same result as CEIL(*argument*).

Examples:

```
DATA;
   X=INT(2.1);
   PUT X;
RUN;
```

produces a value of 2.

```
X=INT(-2.4);
```

produces a value of −2.

```
X=INT(3);
```

produces a value of 3.

```
X=INT(-1.6);
```

produces a value of −1.

INTCK: returns the number of time intervals

```
INTCK(interval,from,to)
```

The INTCK function determines the number of time intervals that occur in a given time span. The result is always an integer value. For example, at the XYZ Company, year-end bonuses are prorated for employees who started work after December 1, the beginning of the company's fiscal year. The payroll department uses the INTCK function to calculate the number of days new employees have worked. (Paid holidays and vacation days are counted; weekends are not counted.) In the following example, the data set EMPLOYEE contains the names and starting dates for each employee.

```
DATA PRORATE;
```

```
SET EMPLOYEE;
W=INTCK('WEEK',STARTING,'30NOV84'D);
D=INTCK('DAY',STARTING,'30NOV84'D);
WDAYS=D-(W*2);
DROP W D;
```

The *interval* must be a character constant or variable whose value is one of those listed below. The *from* and *to* values can be expressed as SAS date, datetime or time values; or as SAS date, datetime, or time literals. The INTCK cannot distinguish date, time, and datetime values from each other, so you must specify the correct type of interval:

- Use date intervals when *from* and *to* contain date values.
- Use datetime intervals when *from* and *to* contain datetime values.
- Use values listed under time intervals when *from* and *to* contain time values.

date intervals	**datetime intervals**	**time intervals**
DAY	DTDAY	HOUR
WEEK	DTWEEK	MINUTE
MONTH	DTMONTH	SECOND
QTR	DTQTR	
YEAR	DTYEAR	

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

The INTCK function counts intervals from fixed interval beginnings, not in multiples of an interval unit from the *from* value. Partial intervals are not counted. For example, WEEK intervals are counted by Sundays rather than seven-day multiples from the *from* argument. YEAR intervals are counted from 01JAN, not in 365-day multiples. The result of

```
INTCK('YEAR',31DEC84'D,'1JAN85'D)
```

is 1, even though only one day has elapsed. The result of

```
INTCK('YEAR','1JAN84'D,'31DEC84'D)
```

is 0, even though 364 days have elapsed. In the first example, an 01JAN date is counted between the *from* and *to* values, so a YEAR interval is added. In the second example 01JAN is not counted between the *from* and *to* values, so an interval is not added.

INTNX: advances a date, time, or datetime value by a given interval

```
INTNX(interval,from,number)
```

The INTNX function generates a SAS date, time, or datetime value that is a given *number* of time intervals from a starting value (*from*). The *interval* must be a character constant or variable whose value is one of those listed under the INTCK description (see above). Note that the INTNX function cannot distinguish date, time, and datetime values from each other, so you must specify the correct type of interval. The *from* argument must be a SAS date, time, or datetime value, or

a SAS date, time, or datetime literal; and *number* gives the number of intervals to use.

For example, the XYZ Company offers its employees medical coverage only after 30 days of full-time employment. The data set called EMPLOYEE contains the names and starting dates for each employee. The following example shows how the personnel office uses the INTNX function to find the date each employee will be eligible for medical coverage.

```
DATA HEALTH;
   SET EMPLOYEE;
   BENEFIT=INTNX('DAY',START,30);
PROC PRINT;
   FORMAT START DATE7. BENEFIT DATE7.;
```

If the *number* argument is 0, the returned value is the first value of the specified *interval* in which the *from* argument falls. For example,

```
X=INTNX('MONTH','05JAN85'D,0);
PUT X=DATE.;
```

produces

```
X=01JAN85.
```

Note that you must specify a SAS date, time, or datetime format to print the result as a readable value; otherwise, results are printed as SAS date, time, or datetime values. See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

JULDATE: returns the Julian date from a SAS date value or literal

```
JULDATE(date)
```

The JULDATE function converts a SAS *date* value or date literal to a numeric value containing the Julian representation of the date. If the date is a 20th century date, there are five digits; the first two are the year, and the next three the day of the year. If the date is not a 20th century date, there are seven digits; the first four are the year, and the next three the day of the year. Thus, 1JAN85 is 85001 in Julian representation; 31DEC1878 is 1878365.

The JULDATE function can be used to convert DATE7.-format dates (for example, 1JAN85) into Julian dates for convenience or consistency. For example, in the example for the DATEJUL function, above, a publishing company kept a data set containing the names of newspaper subscribers and the dates their subscriptions began. The dates were stored as Julian dates, but the DATEJUL function allowed the company to print a report showing the dates in WORDDATE. format. Rather than require the subscription sales department to enter the dates for new subscriptions as Julian dates in order to update the main file, the JULDATE function is used.

```
DATA UPDATE;
   INPUT NAME $ START DATE7.;
   JULIAN=JULDATE(START);
   DROP START;
   CARDS;
data lines
;
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

KURTOSIS: calculates the kurtosis, the 4th moment

```
KURTOSIS(argument,argument,...)
```

The *arguments* for this function must be numeric values. At least four nonmissing values are required. The result is the kurtosis statistic of the nonmissing arguments.

For the example

```
X=KURTOSIS(0,1,0,1);
```

the value returned is -6.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

LAG: lagged values

```
LAGn(argument)
```

A family of LAG functions, named LAG1, LAG2, ..., LAG100, is available for obtaining up to 100 lags of a variable's value. (LAG1 can also be written as LAG.) The LAG function can have either numeric or character arguments. It "remembers" values of the argument from a previous execution of the function.

Each LAG function in a program has its own "stack" of lag values. The stack for a LAG*n* function is initialized with *n* missing values, where *n* is the number of lags (for example, a LAG2 stack is initialized with two missing values). Each time the function executes, the value at the top of the stack is returned, and the current value of the argument is placed at the bottom of the stack. This means that missing values are returned for the first *n* executions of a LAG function, after which the lagged values of the argument are returned.

For example, consider this DATA step:

```
DATA ONE;
   INPUT X @@;
   Y=LAG1(X);
   Z=LAG2(X);
   CARDS;
1 2 3 4
;
```

Data set ONE contains the following values for X, Y, and Z:

```
X   Y   Z
1   .   .
2   1   .
3   2   1
4   3   2
```

The LAG1 function returns one missing value and then the values of X (lagged once); the LAG2 function returns two missing values and then the values of X (lagged twice).

Note: LAGs are only generated when a LAG function executes; they are not automatically generated for each observation. If the LAG function executes con-

ditionally in an IF-THEN statement, only values from those observations meeting the IF condition are used for lagging. The following example demonstrates this.

```
DATA A;
   INPUT Y @@;
   IF _N_>1 THEN X=Y*2+LAG(X);
      ELSE X=Y*2;
   CARDS;
1 2 3 4
;
```

The DATA step causes this error message:

```
MISSING VALUES WERE GENERATED AS A RESULT OF PERFORMING
AN OPERATION ON MISSING VALUES.
```

Data set A contains these values:

```
OBS   Y   X

 1    1   2
 2    2   .
 3    3   .
 4    4   .
```

X is always missing except in the first observation, because LAG1(X) returns a value from the last time the LAG executed, not from the previous observation. The first time the DATA step executes, _N_=1, so the conditions of the IF statement are not met. The ELSE statement executes and X has a value of 2. The first time the LAG function executes, the lag value is a missing value, which causes X to be missing, and all succeeding X values to be missing.

If the argument to LAG is an array name, each array element is lagged separately.

LEFT: left-aligns a character expression

```
LEFT(argument)
```

The LEFT function returns the argument with leading blanks moved to the end of the value. The argument's length does not change. For example, the statements

```
A='   HI THERE';
B=LEFT(A);
```

give *B* the value 'HI THERE '.

LENGTH: gives the length of a character argument

```
LENGTH(argument)
```

The LENGTH function returns the length of the character expression specified by the argument. The result is the position of the right-most nonblank character in the argument. If the argument is missing, LENGTH returns a value of 1. For example,

```
LEN=LENGTH('ABCDEF');
```

gives LEN a value of 6, which is the last character in the argument, F.

LGAMMA: calculates the natural logarithm of the GAMMA function of a value

```
LGAMMA(argument)
```

This function results in the natural logarithm of the GAMMA function of the value of *argument* (see GAMMA). The value of *argument* must be positive.

Example:

```
DATA;
   X=LGAMMA(2);
   PUT X;
RUN;
```

results in a value of 0.

LOG: natural logarithm

```
LOG(argument)
```

The result of this function is the natural (Naperian) logarithm of the value of *argument*. The argument must have a positive value.

The base of natural logarithms is e, approximately 2.71828.

Examples:

```
DATA;
   X=LOG(1);
   PUT X;
RUN;
```

results in a value of 0.

```
X=LOG(10);
```

results in a value of 2.30259.

LOG10: common logarithm

```
LOG10(argument)
```

This function results in the common logarithm (log to the base 10) of the value of the *argument*. The argument must have a positive value.

Examples:

```
X=LOG10(1);
```

results in a value of 0.

```
X=LOG10(10);
```

results in a value of 1.

```
X=LOG10(100);
```

results in a value of 2.

LOG2: logarithm to the base 2

```
LOG2(argument)
```

This function results in the logarithm to the base 2 of the value of *argument*. The argument must have a positive value.

Examples:

```
X=LOG2(2);
```
results in a value of 1.

```
X=LOG2(.5);
```
results in a value of −1.

MAX: returns the largest value

```
MAX(argument,argument, ...)
```

The *arguments* for this function must be numeric values. At least two nonmissing values are required. The result is the largest value among the nonmissing values of the arguments.

For the examples

```
X=MAX(2,6,.);
X=MAX(2,-3,1,-1);
X=MAX(3,.,-3);
```
the values returned are 6, 2, and 3, respectively.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic. Note that the MAX function does not necessarily return the same value as the MAX operator described in "SAS Expressions."

MDY: returns a SAS date value from month, day, and year

```
MDY(month,day,year)
```

The MDY function produces a SAS date value from numeric values that represent *month, day,* and *year*. For example, the SAS statements

```
DATA A;
   M=8;
   D=27;
   Y=47;
   BIRTHDAY=MDY(M,D,Y);
   PUT BIRTHDAY= DATE7.;
```
produce the line

```
BIRTHDAY=27AUG47
```

Note that this result is printed as a SAS date value (number of days from 01JAN60) unless a date format is specified.

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

MEAN: computes the arithmetic mean (average)

```
MEAN(argument,argument, ...)
```

The *arguments* for this function must be numeric values. One or more nonmissing values are required. The function results in the average of the values of the nonmissing arguments.

The examples

```
X=MEAN(2,.,.,6);
X=MEAN(1,2,3,2);
```

return the results 4 and 2, respectively.

You can use an abbreviated argument list preceded by OF.
See "SAS Descriptive Procedures" for a definition of this statistic.

MIN: returns the smallest value

```
MIN(argument,argument, ...)
```

The *arguments* for this function must be numeric values. Two or more nonmissing values are required. The result is the smallest value among the nonmissing values of the arguments.

For the examples

```
X=MIN(2,.,6);
X=MIN(2,-3,1,-1);
X=MIN(0,4);
```

the values returned are 2, −3, and 0, respectively.

See "SAS Descriptive Procedures" for a definition of this statistic. Note that the MIN function does not necessarily return the same value as the MIN operator described in "SAS Expressions."

MINUTE: returns the minute from a SAS time or datetime value or literal

```
MINUTE(time)
MINUTE(datetime)
```

The MINUTE function operates on a SAS *time* or *datetime* value or SAS time or datetime literal to produce a numeric value containing the minute. For example, the statements:

```
DATA A;
   TIME='3:19:24'T;
   M=MINUTE(TIME);
   PUT M=;
```

produce the line

```
M=19
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

MOD: calculates the remainder

```
MOD(argument1,argument2)
```

The result of this function is the remainder when the quotient of *argument1* divided by *argument2* is calculated.

Examples:

```
DATA;
```

```
   X=MOD(6,3);
   PUT X;
RUN;
```

returns the value 0.

```
X=MOD(10,3);
```

returns the value 1.

```
X=MOD(11,3.5);
```

returns the value .5.

```
X=MOD(10,-3);
```

returns the value 1.

MONTH: returns the month from a SAS date value or literal

```
MONTH(date)
```

The MONTH function operates on a SAS *date* value or date literal to produce a numeric value containing the month. For example, the statements

```
DATA A;
   DATE='25DEC79'D;
   M=MONTH(DATE);
   PUT M=;
```

produce the line

```
M=12
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

N: reports the number of nonmissing arguments

```
N(argument,argument, ...)
```

The *arguments* for this function must be numeric values. At least one argument is required. The function returns the number of nonmissing arguments.

For the example

```
X=N(1,0,.,2,5,.);
```

the value returned is 4.

You can use an abbreviated argument list preceded by OF. See "SAS Descriptive Procedures" for a definition of this statistic.

NMISS: reports the number of missing values

```
NMISS(argument,argument,...)
```

The *arguments* for this function must be numeric values. At least one argument is required. The function gives the number of missing values in a string of arguments.

The example

```
X=NMISS(1,10,3,.);
```

returns the value 1. You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

NORMAL: generates a normally distributed pseudo-random variate

```
NORMAL(seed)
```

The *seed* for this function must be a numeric value. It should be either zero (0) or a five-, six-, or seven-digit odd integer. The result is a pseudo-random variate; the variates generated by NORMAL appear to be normally distributed with a mean of 0 and a standard deviation of 1.

The *seed* is used only the first time the function is evaluated. Each subsequent time the function is evaluated, the result of the last evaluation is used in generating the new variate. If *seed* is 0, SAS uses a reading of the time of day from the computer's clock to generate the first variate. Otherwise, the constant specified is used to generate the first variate. An expression

```
X=M+S*NORMAL(seed)
```

can produce variates with mean M and standard deviation S.

The NORMAL function uses a central limit theorem approximation

$$x = \Sigma_{i-1}^{12} u_i - 6$$

where the u_i are produced with the UNIFORM function. (See the RANNOR function for a better generator.)

POISSON: probability values for the Poisson distribution

```
POISSON(lambda,n)
```

where

$0 \leq lambda$ and $0 \leq n$.

This function returns the probability that an observation from a Poisson distribution is less than or equal to *n*. *Lambda* is the value of the mean parameter. A single term of the Poisson distribution may be computed as a difference of two values of the cumulative distribution. If *x*=POISSON(*lambda,n*) then

$$p = \Sigma_{j=0}^{n} e^{-\lambda}(\lambda^j/j!) \quad .$$

Example:

```
DATA;
   P=POISSON(1,2);
   PUT P;
RUN;
```

results in the value .9197.

PROBBETA: probability values from a beta distribution

```
PROBBETA(x,a,b)
```

where

$0 \leq x \leq 1$ and $0 < a,b$.

This function returns probability values from a beta distribution. The *a* and *b* values are the shape parameters of the beta distribution, and *x* is the value at which the distribution is to be evaluated. The density is

$$x^{a-1}(1-x)^{b-1}\Gamma(a+b)/\Gamma(a)\Gamma(b)$$

The incomplete beta function can be obtained from this function by multiplying the beta probability by values of the complete beta function, which can be computed from the GAMMA function.

This function is related to many of the common distributions of statistics and also has applications in analyzing order statistics (see Michael and Schucany 1979).

PROBBNML: probability values from a binomial distribution

```
PROBBNML(p,n,m)
```

where

$$0 \leq p \leq 1,\ 1 \leq n,\ 0 \leq m \leq n \quad .$$

This function returns the probability that an observation from a binomial distribution with parameters *p* and *n* is less than or equal to *m*. The binomial probability parameter is *p*, and *n* is the degree of the binomial distribution. A single term in the binomial distribution can be obtained as the difference of two values of the cumulative binomial distribution.

If

```
P=PROBBNML(p,n,m)
```

then

$$P = \Sigma_{j=0}^{m}\binom{n}{j}p^{j}(1-p)^{n-j} \quad .$$

Example:

```
DATA;
   P=PROBBNML(.5,10,4);
   PUT P;
RUN;
```

results in the value .37695.

PROBCHI: computes probability values for the chi-square distributions

```
PROBCHI(x,df)
```

```
P=1-PROBCHI(31.264,11);
```

returns the value .001. The PROBCHI function computes the probability that a random variable with a chi-square distribution, with *df* degrees of freedom, falls below the *x* value given. This function accepts noninteger degrees of freedom. If the optional third argument is not specified or has the value zero, the central distribution is computed. Otherwise the nc parameter gives the noncentrality value, and the noncentral chi-square distribution is computed. The noncentral chi-square probability function is derived from the functions contributed by Hardison, Quade and Langston (1983).

PROBF: the probability for the F distribution

```
PROBF(x,ndf,ddf)
```

The PROBF function computes the probability that a random variable with an F distribution, with *ndf* numerator degrees of freedom and *ddf* denominator degrees of freedom, falls below the *x* value given. This function accepts noninteger degrees of freedom. If the optional fourth argument is not specified or has the value zero, the central distribution is computed. Otherwise the nc parameter gives the noncentrality parameter to use in computing the noncentral F distribution. The noncentral F probability function is also derived from the functions contributed by Hardison, Quade and Langston. To find the significance level, use

```
1-PROBF(x,ndf,ddf)
```

.

For example,

```
P=1-PROBF(3.32,2,30);
```

returns the value 0.4982954.

PROBGAM: probability values for the gamma distribution

```
PROBGAM(x,eta)
```

The PROBGAM function computes the probability that a random variable with a gamma distribution with shape parameter η falls below the *x* value given. The GAMINV function is the inverse of the PROBGAM function.

The density is

$$x^{\eta-1}e^{-x}/\Gamma(\eta)$$

where

η is the shape parameter.

PROBHYPR: probabilities from a hypergeometric distribution

```
PROBHYPR(nn,k,n,x,or)
```

where

$1 \leq nn$
$1 \leq k \leq nn$
$1 \leq n \leq nn$
$\text{MAX}(0, k + n - nn) \leq x \leq \text{MIN}(k,n)$.

This function returns the probability that an observation from a hypergeometric distribution with total sample *nn*, margins *n* and *k*, and odds ratio *or* is less than or equal to *x*. The *or* argument can be omitted; if it is, the *or* parameter is assumed to be 1.

Examples:

```
DATA;
   X=PROBHYPR(10,5,3,2);
   PUT X;
RUN;
```

results in a value of .9167.

```
X=PROBHYPR(10,5,3,2,1.5);
```

results in a value of .8541.

PROBIT: inverse normal distribution function

```
PROBIT(argument)
```

This function is the inverse of the standard normal cumulative distribution function. The value of *argument* should be between 0 and 1. The result will be truncated, if necessary, to lie between −5 and +5. PROBIT is the function inverse of PROBNORM. If X is a normally distributed random variable with mean 0 and standard deviation 1, then *z* is the probability that X will take on a value less than PROBIT(*z*). Examples:

```
DATA;
   X=PROBIT(.025);
   PUT X;
RUN;
```

results in a value of −1.96.

PROBNEGB: probability values for the negative binomial distribution

```
PROBNEGB(p,n,m)
```

where

$0 \leq p \leq 1$, $0 < n$, and $0 \leq m$.

This function returns the probability that an observation from a negative binomial distribution with parameters *p* and *n* is less than or equal to *m*. The binomial probability parameter is *p* and *n* is the degree of the negative binomial distribution. The value of a single term in the negative binomial distribution can be obtained by a difference of two values of the cumulative distribution.

If X=PROBNEGB(*p*,*n*,*m*) then:

$$X = \Sigma_{j=0}^{m}(1-p)^j p^n (n + j_j - 1)$$

Example:

```
DATA;
   X=PROBNEGB(.5,2,1);
   PUT X;
RUN;
```

results in a value of .5.

PROBNORM: computes probabilities for normal distributions

```
PROBNORM(x)
```

The PROBNORM function computes the probability that a random variable with a normal (0,1) distribution falls below the *x* value given. This function is equivalent to

```
.5+ERF(X/SQRT(2))/2)    .
```

The PROBIT function is the function inverse of PROBNORM.
Examples:

```
DATA;
   X=PROBNORM(0);
   PUT X;
RUN;
```

results in a value of .5.

```
X=PROBNORM(1.96);
```

results in a value of .975.

PROBT: the probability for the *t* distribution function

```
PROBT(x,df)
```

The PROBT function computes the probability that a random variable with a student's *t* distribution with *df* degrees of freedom falls below the *x* value given. This function accepts noninteger degrees of freedom. The third argument is optional and if omitted or zero, the central t distribution is computed. Otherwise the third argument is the noncentrality value for the noncentral t distribution. The noncentral t probability function is derived from the functions contributed by Hardison, Quade and Langston(1983). For a two-tailed test, compute the significance level by

(1−PROBT(ABS(x),DF))*2 .

Example:

```
DATA;
   X=PROBT(.9,5);
   PUT X;
RUN;
```

results in a value of 0.795.

PUT: specifies an output format for a value

```
PUT(argument,format)
```

This function allows you to "write" *argument* with the format specified in the second argument. The format must be the same TYPE as the first argument. The result of the PUT function is **always** a character string. This is useful for converting a numeric value to a character value, or for changing the character format assigned to a variable or value.

For example, this statement converts the values of a numeric variable CC containing completion codes into the three-character hex representation of the codes

```
CCHEX=PUT(CC,HEX3.);
```

CCHEX's value is the same as the characters that would be written with the statement

```
PUT CC HEX3.;
```

If the first argument is a numeric variable, the resulting string is right-aligned. If the first argument is a character variable, the result is left-aligned.

QTR: returns the quarter from a SAS date value or literal

```
QTR(date)
```

The QTR function return a value of 1, 2, 3, or 4 from a SAS *date* value or date literal to indicate the quarter in which a date value falls.

Examples:

```
QTR (3005) = 1
QTR ('20JAN82'D) = 1
```

RANBIN: generates an observation from a binomial distribution with parameters *n* and *p*

```
RANBIN(seed,n,p)
```

where

$n>0$ integer, and $0<p<1$.

For any numeric *seed* value (see **Notes on Random Number Functions** earlier in this chapter for a complete discussion of *seed*) the RANBIN function generates an observation of a binomial variate with mean np and variance $np(1-p)$. If $n\leq 50$, the inverse transform method is applied to a RANUNI uniform deviate. If $n>50$, the normal approximation to the binomial distribution is used. In this case, the normal deviate is generated using the Box-Muller transformation of RANUNI uniform deviates.

The CALL RANBIN subroutine, an alternative to the RANBIN function, gives you greater control of the seed and random number streams. For more information on the use of CALL RANBIN, see **Notes on Random Number Functions** earlier in this chapter.

RANCAU: generates a Cauchy deviate

```
RANCAU(seed)
```

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANCAU function generates an observation of a Cauchy random variable with location parameter 0 and scale parameter 1. An acceptance-rejection procedure and RANUNI uniform deviates are used for generation. The technique relies on the fact that if u and v are independent uniform $(-1/2, 1/2)$ variables and $u^2+v^2\leq 1/4$ then u/v is a Cauchy deviate.

If

```
X=ALPHA+BETA*RANCAU(SEED)
```

then X is a *Cauchy variate* with location parameter ALPHA and scale parameter BETA.

The CALL RANCAU subroutine, an alternative to the RANCAU function, gives you greater control of the seed and random number streams. For details on the use of CALL RANCAU, see **Notes on Random Number Functions**.

RANEXP: generates an exponential deviate

```
RANEXP(seed)
```

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANEXP function generates an observation of an exponential variate with parameter 1. The inverse transform method applied to a RANUNI uniform deviate is used for generation.

If

```
X=RANEXP(SEED)/LAMBDA
```

then X is an *exponential variate* with parameter LAMBDA.

If

```
X=ALPHA-BETA*LOG(RANEXP(SEED))
```

then X is an an *extreme value variate* with location parameter ALPHA and scale parameter BETA.

If

```
X=FLOOR(-RANEXP(SEED)/LOG(P))
```

then X is a *geometric variate* with parameter P.

The CALL RANEXP subroutine, an alternative to the RANEXP function, gives you greater control of the seed and random number streams. For details on the use of CALL RANEXP, see **Notes on Random Number Functions** earlier in this chapter.

RANGAM: generates an observation from a gamma distribution with shape parameter *alpha*

```
RANGAM(seed,alpha)
```

where

alpha>0 .

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANGAM function generates an observation from a gamma distribution with density function

$$f(x) = x^{\alpha-1}e^{-x}/\Gamma(\alpha)$$

where $\alpha>0$, and $x>0$.

A combination of techniques is used in generating an observation. For a noninteger *alpha*, two independent gamma variates are generated: one with parameter INT(*alpha*), the integer part of *alpha*, and the other with parameter *alpha*−INT(*alpha*). The sum of the gamma variates has the desired distribution. For integer *alpha*, only the first of these need be generated. The inverse transformation of a RANUNI uniform deviate is used to generate the gamma variate with parameter INT(*alpha*) and an acceptance-rejection method is used to generate the other gamma variate (Fishman 1978). To expedite execution, internal variables used in generation are calculated only on initial calls (that is, with each new *alpha*).

If

```
X=BETA*RANGAM(SEED,ALPHA)
```

then X is a *gamma variate* with shape parameter ALPHA and scale parameter BETA.

If 2*ALPHA is an integer, and

```
X=2*RANGAM(SEED,ALPHA)
```

then X is a *chi-square variate* with 2*ALPHA degrees of freedom.

If

```
X=BETA*RANGAM(SEED,N)
```

where N is a positive integer, then X is an *Erlang variate*. It has the distribution of the sum of N independent exponential variates whose means are BETA.

If

```
Y1=RANGAM(SEED,ALPHA)
Y2=RANGAM(SEED,BETA)
X=Y1/(Y1+Y2)
```

then X is a *beta variate* with parameters ALPHA and BETA, and density function

$$f(x) = \Gamma(\alpha + \beta)/\Gamma(\alpha)\Gamma(\beta)x^{\alpha-1}(1-x)^{\beta-1}$$

where

$0 \leq x \leq 1$ and $\alpha, \beta > 0$.

The CALL RANGAM subroutine, an alternative to the RANGAM function, gives you greater control of the seed and random number streams. For details on CALL RANGAM, see **Notes on Random Number Functions** earlier in this chapter.

RANGE: reports the range of values

```
RANGE(argument,argument, ...)
```

The *arguments* for this function must be numeric values. The function requires two or more arguments. The result is the range of the values of the nonmissing arguments.

For example

```
X=RANGE(2,6,3);
```

or

```
X=RANGE(2,6,3,.);
```

return the result 4.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

RANNOR: generates a normal deviate

```
RANNOR(seed)
```

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANNOR function generates an observation of a normal random variable with mean 0 and variance 1. The Box-Muller transformation of RANUNI uniform deviates is used for generation.

If

```
X=MU+SQRT(SIGMASQ)*RANNOR(SEED)
```

then X is a *normal variate* with mean MU and variance SIGMASQ.

If

```
X=EXP(MU+SQRT(SIGMASQ)*RANNOR(SEED))
```

then X is a *lognormal variate* with mean

```
EXP(MU+SIGMASQ/2)
```

and variance

```
EXP(2*MU+SIGMASQ)*(EXP(SIGMASQ)-1)    .
```

The CALL RANNOR subroutine, an alternative to the RANNOR function, gives you greater control of the seed and random number streams. For details on the use of CALL RANNOR, see **Notes on Random Number Functions**, earlier in this chapter.

RANPOI: generates an observation from a Poisson distribution with parameter *lambda*

```
RANPOI(seed,lambda)
```

where

lambda>0 .

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANPOI function generates an observation of a Poisson variate. The inverse transform method applied to a RANUNI uniform deviate is the generating technique. The method for inverting the cumulative distribution function varies with the value of the parameter *lambda*. For a noninteger *lambda*, two independent Poisson variates are generated: one with parameter INT(*lambda*), the integer part of *lambda*; the other with parameter *lambda*−INT(*lambda*). The sum of the Poisson variates is returned (Fishman 1976). For integer *lambda*, only the first of these need be generated. To expedite execution, internal variables used in generation are calculated only on initial calls (that is, with each new *lambda*).

The CALL RANPOI subroutine, an alternative to the RANPOI function, gives you greater control of the seed and random number streams. For details on the use of CALL RANPOI, see **Notes on Random Number Functions** earlier in this chapter.

RANTBL: generates deviates from a tabled probability mass function

```
RANTBL(seed,p1,...pi,...,pn)
```

where

$0 \leq p_i \leq 1$ for $0 < i \leq n$.

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANTBL function returns an observation generated from the probability mass function defined by p_1 through p_n. In particular

RANTBL = 1 with probability p_1
2 with probability p_2
.
.
.
n with probability p_n .

The inverse transform method applied to a RANUNI uniform deviate is used in generation.

Note: if you execute

```
X=RANTBL(SEED,P1,...,Pn);
IF X=1 THEN X=M1;
ELSE IF X=2 THEN X=M2;
     .           .
     .           .
     .           .
ELSE IF X=n THEN X=Mn;
```

then *X* takes the values M1 through M*n* with probabilities P1 through P*n*, respectively.

The CALL RANTBL subroutine, an alternative to the RANTBL function, gives you greater control of the seed and random number streams. For details on the use of CALL RANPOI, see **Notes on Random Number Functions** earlier in this chapter.

RANTRI: generates an observation from the triangular distribution with parameter *h*

```
RANTRI(seed,h)
```

where

$0<h<1$.

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANTRI function returns an observation generated from the triangular distribution with density function

$$f(x) = 2x/h$$

if

$$0 \leq x \leq h$$

and

$$f(x) = 2(1 - x)/(1 - h)$$

if

$h<x\leq 1$.

The inverse transform method applied to a RANUNI uniform deviate is used for generation.

If you execute

```
X=(B-A)*RANTRI(SEED,(C-A)/(B-A))+A;
```

then *X* has a triangular distribution on the interval [A,B] with mode C ε [A,B].

The CALL RANTRI subroutine, an alternative to the RANTRI function, gives you greater control of the seed and random number streams. For details on the use of CALL RANTRI, see **Notes on Random Number Functions** earlier in this chapter.

RANUNI: generates a uniform deviate

```
RANUNI(seed)
```

For any numeric *seed* value (see **Notes on Random Number Functions** for a complete discussion of *seed*) the RANUNI function returns a number generated from the uniform distribution on the interval (0,1) using a prime modulus multiplicative generator with modulus $2^{31}-1$ and multiplier 397204094 (Fishman and Moore 1982). The *seed* must be a numeric constant less than $2^{31}-1$.

REPEAT: repeats a character expression

```
REPEAT(argument1,n)
```

The REPEAT function returns a character value consisting of the first argument repeated *n* times. Thus, the first argument appears *n*+*1* times in the result. For example, the statement

```
X=REPEAT('ONE',2);
```

gives the value ONEONEONE.

REVERSE: reverses a character expression

```
REVERSE(argument)
```

The REVERSE function returns the argument's characters in reverse order. Trailing blanks are retained and therefore the argument's length does not change. For example,

```
BACKWARD=REVERSE('abc  ');
```

gives BACKWARD the value ' cba'.

RIGHT: right-aligns a character expression

```
RIGHT(argument)
```

The RIGHT function returns the argument with trailing blanks moved to the beginning of the value. The character expression's length does not change. For example, the statements

```
A='HI THERE   ';
B=RIGHT(A);
```

give B the value ' HI THERE'.

ROUND: rounds a value to nearest roundoff unit

```
ROUND(argument,roundoffunit)
```

The ROUND function rounds a value to the nearest roundoff unit.
Examples:

```
DATA;
   X=ROUND(223.456,1);
   PUT X;
RUN;
```

results in a value of 223.

```
X=ROUND(223.456,.01);
```

results in a value of 223.46.

```
X=ROUND(223.456,100);
```

results in a value of 200.

```
X=ROUND(223.456);
```

results in a value of 223.

The value of the *roundoffunit* must be greater than zero. If the *roundoffunit* is omitted, a value of 1 is used and *argument* is rounded to the nearest integer.

SASVER: returns the current version of SAS

```
SASVER()
```

The SASVER function returns the current version of SAS. For example,

```
VERSION=SASVER();
PUT 'VERSION=' VERSION:
```

produces the line

VERSION=5.08

Operating systems: CMS, OS, VM/PC, and VSE.

SCAN: returns a given word from a character expression

```
SCAN(argument1,n,delimiters)
SCAN(argument1,n)
```

SCAN separates the first argument, a character expression, into "words," and returns the *n*th word. The *delimiters* are characters defined as separators between words. If the *delimiters* argument is not specified, the following characters are default *delimiters*:

blank . < (+ | & ! $ *) ; ¬ − / , % ¦ ¢

The statements

```
ARG='ABC.DEF(X=Y)';
WORD=SCAN(ARG,3);
```

give WORD the value X=Y, the third "word" in the character expression, ARG. The following statement shows another way to accomplish the same thing.

WORD=SCAN('AB?DEF?X=Y?',3,'?');

Notice that in the above statement the first argument is a character literal, rather than a variable, and that the third argument specifies a question mark (?) as the delimiter. The character chosen for the delimiter argument must be enclosed in quotes.

Leading delimiters before the first word have no effect. If there are two or more contiguous delimiters, they are treated as one. If there are fewer than *n* words in the first argument, SCAN returns a blank value.

SECOND: returns the second from a SAS time or datetime value or literal

```
SECOND(time)
SECOND(datetime)
```

The SECOND function operates on a SAS *time* or *datetime* value or a SAS time or datetime literal to produce a numeric value containing the seconds part of the value. For example, the statements

```
DATA A;
```

```
TIME='3:19:24'DT;
S=SECOND(TIME);
PUT S=;
```

produce

S=24

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

SIN: trigonometric sine

```
SIN(argument)
```

The *argument* for this function must be a numeric value. The function results in the sine of the value of argument. The value of the argument is assumed to be in radians.

The examples

```
X=SIN(.5);
X=SIN(0);
X=SIN(3.14159/4);
```

return 0.4794255, 0, and 0.7071063, respectively.

SINH: hyperbolic sine

```
SINH(argument)
```

The *argument* for this function must be a numeric value. The result is the hyperbolic sine of the value of the *argument*. It is equivalent to:

```
(EXP(argument)-EXP(-argument))/2
```

For example,

```
X=SINH(0);
```

returns the result 0.

SKEWNESS: gives the skewness

```
SKEWNESS( argument, argument, argument,...)
```

The *arguments* for this function must be numeric values. More than two arguments are required. The result of this function is a measure of the skewness of the nonmissing argument values.

For the examples

```
X=SKEWNESS(0,1,1); or X=SKEWNESS(0,.,1,1);
```

and

```
X=SKEWNESS(2,4,6,3,1);
```

the values returned are −1.73205 and 0.5901287, respectively. You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

STD: calculates the standard deviation

```
STD(argument,argument, ...)
```

The *arguments* for this function must be numeric values. At least two arguments must be provided. The function gives the standard deviation of the values of the nonmissing arguments.

The examples

```
X=STD(2,6); or X=STD(2,6,.);
```

and

```
X=STD(2,4,6,3,1);
```

return the values 2.828427 and 1.923538, respectively.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

STDERR: calculates the standard error of the mean

```
STDERR(argument,argument,...)
```

The *arguments* for this function must be numeric values. At least two nonmissing values are required. The function results in the standard error of the mean of the nonmissing values of the arguments.

For example,

```
X=STDERR(2,6,3,4);
```

or

```
X=STDERR(2,6,.,3,4);
```

returns the result 0.8539126.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

STFIPS: converts state postal code to FIPS state code

```
STFIPS(postalcode)
```

The STFIPS function takes a two-character state postal code (enclosed in quotes) and converts it to the corresponding numeric FIPS state code.

Example:

```
FIPS=STFIPS ('NC');
PUT FIPS=;
```

produces

```
FIPS=37
```

STNAME: converts state postal code to state name (all uppercase)

```
STNAME(postalcode)
```

The STNAME function takes a two-character state postal code (enclosed in quotes) and returns the corresponding 20-character state name in uppercase.

For example,

```
STATE=STNAME('NC');
PUT STATE=;
```

produces

```
STATE=NORTH CAROLINA
```

STNAMEL: converts state postal code to state name in upper- and lowercase

```
STNAMEL(postalcode)
```

The STNAMEL function takes a two-character state postal code (enclosed in quotes) and returns the corresponding twenty-character state name in upper- and lowercase. For example,

```
STATE=STNAMEL('NC');
PUT 'STATE=' STATE;
```

results in the value

```
STATE=North Carolina
```

SUBSTR: a dual function: substring and pseudo-variable for character insertion

The SUBSTR function serves two different purposes. They are described below.

substring

```
SUBSTR(argument1,position,n)
```

When the SUBSTR function is used on the right side of an equal sign (=), it extracts from *argument1* a substring that is *n* characters long, beginning with the character specified by *position*. If *n* is omitted, the substring consists of the remainder of *argument,* beginning with the character specified by *position*.

For example, the statements

```
DATA A;
   DATE='06MAY85';
   MONTH=SUBSTR(DATE,3,3);
   YEAR=SUBSTR(DATE,6,2);
```

produce the value MAY for MONTH and the value 85 for YEAR.

When the SUBSTR function is on the right side of an equal sign, any of the arguments can be literal, derived, or variable character expressions.

pseudo-variable for character insertion

When SUBSTR is used on the left side of an equal sign, it serves as a pseudo-variable function, replacing the contents of a character value.

```
SUBSTR(argument1,position,n)=x;
```

The value of the variable or constant on the right side of the equal sign is placed into the character variable specified by the first argument, starting with the character specified by *position*, and replacing the number of characters specified by *n*. Notice that the first argument **must** be a character variable. This restriction is an exception and applies only when SUBSTR is used as a pseudo-variable function. For example, the statements

```
A='KIDNAP';
SUBSTR(A,1,3)='CAT';
```

give A the value CATNAP.

If the *n* argument is omitted, characters in the first argument are replaced from *position* to the end of the first argument. In the following example, SUBSTR replaces characters from 'N', the position specified by the second argument, to the end of the character expression

```
A='CATNAP';
SUBSTR(A,4)='TY';
```

giving A the value CATTY.

SUM: calculates the sum of the arguments

```
SUM(argument,argument, ...)
```

The *arguments* for this function must be numeric values. Two or more arguments are required. The result of the SUM function is the sum of the arguments.

For example,

```
X=SUM(4,9,3,8);
```

returns the value 24.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

SYMGET: returns the value of a macro variable

```
SYMGET(variable)
```

The *variable* specified in the SYMGET function is a macro variable, or a variable whose values are names of macro variables. If you want to use a macro variable as the argument, it must be enclosed in single quotes; for example, SYMGET('MACNAME'). Macro variables are discussed in the chapter, "SAS Macro Language." The value of a macro variable is a character string, so the SYMGET function returns a character value. If the function is used in an assignment statement, the resulting variable is a character variable.

If the value of the macro variable is longer than the resulting variable, SAS truncates the returned value on the right.

In these statements, macro variables SYM1, SYM2, and SYM3 are created, and their values are retrieved with the SYMGET function in a DATA step.

```
%LET SYM1=AAA;
%LET SYM2=BBB;
%LET SYM3=CCC;
DATA NEW;
   INPUT CODE $ @@;
   X=SYMGET(CODE);
   PUT CODE=X=;
   CARDS;
SYM2 SYM3 SYM1 SYM1 SYM3
;
```

These lines are produced:

```
CODE=SYM2 X=BBB
CODE=SYM3 X=CCC
CODE=SYM1 X=AAA
CODE=SYM1 X=AAA
CODE=SYM3 X=CCC
```

The SYMGET function is especially useful with the CALL SYMPUT statement. Both are described in "SAS Macro Language."

SYSPARM: returns the system parameter string

```
SYSPARM( )
```

The SYSPARM function lets you access a character string specified with the SYSPARM= system option in the job control for your job or in an OPTIONS statement.

Example:

```
OPTIONS SYSPARM='YES';
DATA A;
   IF SYSPARM( )='YES' THEN DO;
   .
   .
   .
```

See "SAS System Options" for a discussion of the SYSPARM= option.

If the SYSPARM= option is not specified, the SYSPARM function returns a null string.

Operating systems: CMS, OS, VM/PC, and VSE.

TAN: trigonometric tangent

```
TAN(argument)
```

The *argument* for this function must be a numeric value. The function results in the tangent of the value of the argument. The value of the argument is assumed to be in radians; it may not be an odd multiple of $\pi/2$.

The examples

```
X=TAN(.5);
X=TAN(0);
X=TAN(3.14159/3);
```

return the values 0.5463025, 0, and 1.732047, respectively.

TANH: hyperbolic tangent

```
TANH(argument)
```

The *argument* for this function must be a numeric value. The result is the hyperbolic tangent of the value of the argument. It is equivalent to

```
(EXP(argument)-EXP(-argument))
```

divided by

```
(EXP(argument)+EXP(-argument)) .
```

For the examples

```
X=TANH(-.5);
```

```
X=TANH(0);
```

the values returned are −0.462117 and 0.

TIME: returns the current time of day

```
TIME()
```

The TIME function produces the current time of day as a SAS time value. For example, the statements

```
DATA A;
   CURRENT=TIME();
   PUT CURRENT= TIME.;
```

if executed at exactly 2:32 p.m., produce the line

```
CURRENT=14:32:00
```

Refer also to the example for the HOUR function; it demonstrates the TIME function.

Note that unless a TIME. format is specified, the result is printed as a SAS time value (the number of seconds since 12:00 midnight).

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

TIMEPART: extracts the time part of a SAS datetime value or literal

```
TIMEPART(datetime)
```

The TIMEPART function converts a SAS *datetime* value or datetime literal into just the time part. For positive dates, the statement

```
TIME=TIMEPART(datetime);
```

is equivalent to

```
TIME=MOD(datetime,24*60*60);
```

For example, at 10:40:17 a.m., these statements

```
DATIM=DATETIME();
TIME=TIMEPART(DATIM);
PUT TIME= TIME.;
```

result in this line

```
TIME=10:40:17
```

Note that the result is printed as a SAS time value (number of seconds since 12:00 midnight) unless a SAS time format is specified.

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

TODAY: returns the current date as a SAS date value

```
TODAY()
```

The TODAY function, which can also be written

```
DATE()
```

produces the current date as a SAS date value. For example, executing the statements below on January 20, 1985

```
DATA A;
   CURRENT=TODAY();
   PUT CURRENT= DATE7.;
```

produces the line:

```
CURRENT=20JAN85
```

Note that unless you specify a date format, the result is printed as a SAS date value (the number of days since 01JAN60).

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

TRANSLATE: replaces specific characters in a character expression

```
TRANSLATE(argument1,to,from,...,to,from)
```

Use the TRANSLATE function to replace any occurrence of a character(s) in the first *argument*, specified by *from*, with the character(s) specifed by *to*. For example, the statement:

```
X=TRANSLATE('XYZW','AB','VW');
```

gives X the value XYZB. TRANSLATE searches XYZW for the *from* characters, V and W, and finds a W. In this case, since W is the second character of the value VW, the second character of the value AB replaces W. Thus X's value becomes XYZB.

If the *to* value is shorter than the *from* value, the *to* value is padded with blanks. The result of this padding is that the "extra" *from* characters are converted to blanks. If the *to* value is longer than the *from* value, the *to* value is truncated on the right. Multiple *to* and *from* pairs can be specified.[4]

TRIM: removes trailing blanks from a character expression

```
TRIM(argument)
```

The TRIM function returns the *argument* with trailing blanks removed. TRIM can be useful when concatenating, because concatenation does not remove trailing blanks. For example, consider these statements:

```
DATA A;
   INPUT FIRST $ 1-10 LAST $ 12-25;
   NAME=TRIM(FIRST)||' '||TRIM(LAST);
   CARDS;
JOHN       SMITH
;
```

NAME's value is 'JOHN SMITH'. (The quotes surrounding the blank between the concatenation symbols separate FIRST and LAST with a single blank.) If FIRST and LAST are concatenated without the TRIM function, as in the following statement

```
NAME=FIRST||LAST;
```

NAME's value is 'JOHN SMITH ', with six trailing blanks retained from FIRST and nine trailing blanks retained from LAST.

The length of the receiving variable is the same as that of the argument. That is, the variable returned by the TRIM function is padded with blanks to expand it to the argument's length. This is a problem only when you are doing nothing but trimming the argument. As you can see from the example above, when you are performing an additional operation such as concatenation, the length is not a problem. The following example illustrates this point:

```
FIRST='John     ';
LAST='Smith';
M=FIRST||LAST;
PUT M;
L=TRIM(FIRST)||' '||LAST;

PUT L;
MM=LENGTH(M);
LL=LENGTH(L);
PUT MM LL;
```

The output produced is

```
M=John     Smith
L=John Smith
MM=13   LL=10
```

TSO: invokes a TSO command and returns the status code

```
TSO(command)
```

The TSO function invokes a TSO command that executes during the SAS job. The *command* is a character string enclosed in quotes corresponding to a TSO command. The function returns the status code after the TSO command has executed.

Example:

```
RC=TSO('ALLOC F(STUDY) DA(MY.LIBRARY)')
```

The variable RC contains the status code returned after the TSO ALLOC command specified in the TSO function executes.

Operating system: OS

UNIFORM: generates a pseudo-random variate uniformly distributed on the interval (0,1)

```
UNIFORM(seed)
```

The result of this function is a pseudo-random variate; the variates generated by UNIFORM appear to be uniformly distributed on the interval (0,1).

Seed must be a constant; either 0 or a five-, six-, or seven-digit odd integer. The seed is used only the first time the SAS System evaluates the function. Each subsequent time the function is evaluated, the result of the last evaluation is used in generating the new variate. If the seed is 0, SAS uses a reading of the time of day from the computer's clock to generate the first variate. Otherwise, the constant specified is used to generate the first variate.

UNIFORM is a multiplicative congruential generator with multiplier 16807, modulus 2^{31}, and a 64-value shuffle table to remove autocorrelation. The method used is documented in Lewis, Goodman, and Miller 1969; and Kennedy and Gentle 1980.

(See the RANUNI function for a better generator.)

UPCASE: converts all characters in the argument to uppercase

```
UPCASE(argument)
```

The UPCASE function converts all characters in the argument to uppercase. For example,

```
NAME=UPCASE('John B. Smith');
```

gives NAME the value 'JOHN B. SMITH'.

VERIFY: returns the position of the first character that is unique to the first argument.

```
VERIFY(argument1, argument2,...,argumentn)
```

The VERIFY function returns the position of the first character in *argument1* that is not present in any of the other arguments. If all characters in the first argument are found in at least one of the arguments, VERIFY returns a 0. For example, consider these statements:

```
DATA A;
   CHECK='ABCDE';
   INPUT GRADE $ 1;
   X=VERIFY(GRADE,CHECK);
   IF X NE 0 THEN PUT 'INVALID GRADE VALUE';
```

These statements read a character value from data lines and check that its characters are the letters A through E. If VERIFY finds any other characters, it prints a message on the log.

USS: calculates the uncorrected sum of squares

```
USS(argument,argument,...)
```

The *arguments* for this function must be numeric values. At least two arguments are required. The result of the USS function is the uncorrected sum of squares of the nonmissing arguments.

For the examples

```
X=USS(4,2,3.5,6);
```

and

```
X=USS(4,2,3.5,6,.);
```

the value returned is 68.25.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

VAR: calculates the variance

```
VAR(argument,argument...)
```

The *arguments* for this function must be numeric values. At least two arguments are required. The VAR function calculates the variance of the nonmissing values of the arguments.

For the examples

```
X=VAR(4,2,3.5,6);
```

and

```
X=VAR(4,2,3.5,6,.);
```

the value returned is 2.729167.

You can use an abbreviated argument list preceded by OF.

See "SAS Descriptive Procedures" for a definition of this statistic.

WEEKDAY: returns day week from SAS date or literal

```
WEEKDAY(date)
```

The WEEKDAY function converts a SAS *date* value or date literal into a number representing the day of the week, where 1=Sunday, 2=Monday, ..., 7=Saturday.

Example:

```
WEEKDAY('20JAN85'D) = 1
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

YEAR: returns the year from a SAS date value or literal

```
YEAR(date)
```

The YEAR function operates on a SAS *date* value or date literal to produce a four-digit numeric value containing the year. For example, the statements

```
DATA A;
   DATE=MDY(12,25,84);
   Y=YEAR(DATE);
   PUT Y=;
```

produce

```
Y=1984
```

See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats" for a discussion of date and time values, informats, formats, and literals.

YYQ: returns a SAS date value from the year and quarter

```
YYQ(year,quarter)
```

The YYQ function returns a SAS date value corresponding to the first day of the specified quarter. The *year* value may be either a two- or four-digit year, the *quar-*

ter value must be either 1, 2, 3, or 4. If either the year or quarter is missing, or if the quarter value is not 1, 2, 3, or 4, the result is missing.

For example, the statements

```
DATA DATES;
   DV=YYQ(85,3);
   PUT DV= DATE7.;
```

produce the line

```
DV=01JUL85
```

The result is printed as a SAS date value (number of days since 01JAN60) unless you specify a date format. See **Using SAS Date, Time, and Datetime Informats and Formats** in "SAS Informats and Formats."

ZIPFIPS: converts ZIP code to FIPS state code

```
ZIPFIPS(zipcode)
```

The ZIPFIPS function takes a five-character ZIP code (enclosed in quotes) and returns the corresponding numeric FIPS state code. For example, the statements

```
FIPS=ZIPFIPS('27511');
PUT 'FIPS=' FIPS;
```

produce

```
FIPS=37
```

ZIPNAME: converts ZIP codes to state names (all uppercase)

```
ZIPNAME(zipcode)
```

The ZIPNAME function takes a five-character ZIP code (enclosed in quotes) and returns the corresponding 20-character state name in uppercase. For example, the statements

```
STATE=ZIPNAME('27511');
PUT 'STATE=' STATE;
```

result in the value

```
STATE=NORTH CAROLINA
```

ZIPNAMEL: converts ZIP codes to state names in upper- and lowercase

```
ZIPNAMEL(zipcode)
```

The ZIPNAMEL function takes a five-character ZIP code (enclosed in quotes) and returns the corresponding 20-character state name in upper- and lowercase. For example, the statements

```
STATE=ZIPNAMEL('27511');
PUT 'STATE=' STATE;
```

produce

```
STATE=North Carolina.
```

ZIPSTATE: converts ZIP codes to state postal codes

```
ZIPSTATE(zipcode)
```

The ZIPSTATE function takes a five-character ZIP code (enclosed in quotes) and returns the corresponding two-character state postal code. For example, the statements

```
ST=ZIPSTATE('27511');
PUT 'ST=' ST;
```

produce

```
ST=NC
```

REFERENCE

Hardison, C.D., Quade, D., and Langston, R.D. (1983) "Nine Functions for Probability Distributions," *SUGI Supplemental Library User's Guide, 1983 Edition*. Cary, NC: SAS Institute Inc.

NOTES

1. **AOS/VS, PRIMOS, and VMS**: The COLLATE function returns characters from the ASCII collating sequence, which has values from 0 through 127. The returned string ends, or truncates, with the character having a value of 127 if you request a string length that would contain characters exceeding this value.

   CMS, OS, and VSE: COLLATE returns characters from the IBM 360/370 (EBCDIC) collating sequence, which has values from 0 through 255. The returned string ends, or truncates, with the character having a value of 255 if you request a string length that would contain characters exceeding this value.

2. **AOS/VS, PRIMOS, and VMS**: The ASCII collating sequence string is padded with blanks to a length of 200.

3. **CMS, OS, and VSE**: Only one-dimensional, explicitly subscripted arrays can be used with the DIM function. The form DIM*n* is not valid.

4. **AOS/VS, PRIMOS, and VMS**: An uneven number of arguments must be specified, that is, *to* and *from* values must be in pairs.

5. **CMS, OS, and VSE**: If one or more *from* values are missing (indicated by a null argument ',,'), the IBM System 360/370 collating sequence is assumed, in ascending order from '00'X to 'FF'X. Each new sequence starts in the collating sequence where the last missing sequence stops. If the *to* value is so long that it causes the collating sequence to exceed the highest possible character, 'FF'X, the *to* value is truncated and the next missing sequence starts over at '00'X. This feature is useful for substitution of nonprintable characters. For example,

```
LENGTH V1 V2 $4;
INPUT V1 $HEX8.;
V2=TRANSLATE(V1,REPEAT('.',63),);
PUT V1=$HEX16. V2=$CHAR4.;
CARDS;
004C0300
40004000
7A7A027A
;
```

Chapter 7

DATA Step Applications

This chapter describes some common DATA step applications. In the first section, examples illustrate DO-group processing, arrays, and selected functions. The next part gives details on using the SET, MERGE, and UPDATE statements. The third section describes how to write customized reports.

PROGRAMMING IN THE DATA STEP

This section gives examples of using the DO statement, the ARRAY statement, and selected functions.

DO-Group Processing

Suppose you have a file of prospective clients containing three different kinds of records: name records, address records, and comment records. Each record type is identified by a code in the first two bytes of the record: .n identifies name records, .a identifies address records, and .c identifies comment records. The number of times each record type occurs varies in each group of name, address, and comment records. The data below are stored in a file described in the control language for the job or session by fileref IN. See the appendix "Operating System Notes" for information on defining files under your operating system.

```
.n Ann Lincoln
.a 101 S. Main St.
.a Anytown, USA
.c good prospect
.c called 01MAY84
.c called 04AUG84
.n Mary Scott
.a 1 Castle Street
.a Scotland
.c referred by J. B. King
.c personal visit 01MAY85
.n John B. Doe
.a P.O. Box 3939
.a University of Anystate
.a Local, USA
.n Mary B. Smith
.n Mark E. Smith
.a 123 Home Street
.a Hometown, USA
```

You want to create a temporary SAS data set and an external (non-SAS) file that contain only the name records from the original file. The SAS data set should contain one variable, NAME; the external file should contain a copy of the name records. The external file is described in the control language for the job or session by fileref OUT.

The following three DATA steps each use a statement from the DO/END family of statements to produce the SAS data set and external file.

IF/THEN/DO To create this job with IF/THEN/DO statements, code:

```
DATA NEW;
   INFILE IN;
   FILE OUT;
   INPUT CODE $2. @;
   IF CODE='.n' THEN DO;
      PUT _INFILE_;
      INPUT NAME $CHAR25.;
      OUTPUT;
      END;
   KEEP NAME;
```

The DATA step above illustrates these SAS features:

- creating a SAS data set and an external file in the same step
- trailing at sign (@) in an INPUT statement
- PUT _INFILE_ for copying input records
- KEEP statement.

The first INPUT statement brings a record into the input buffer of the DATA step and reads the value for CODE from the first two bytes of the record. The trailing at sign holds the record in the buffer. The DO group is executed only when the value of CODE is .n. In that case:

1. The PUT _INFILE_ statement writes the current record from the input buffer to the file described by fileref OUT.
2. The second INPUT statement reads the value of NAME from the record in the buffer.
3. SAS outputs the new observation to the SAS data set.

The KEEP statement appears at the end of the DATA step. It could appear anywhere in the step and have the same effect of specifying the variable to be output to the SAS data set being created.

The lines written to the external file and SAS data set NEW are shown in **Output 7.1** and **Output 7.2**.

Output 7.1 External File with Name Records

```
.n Ann Lincoln
.n Mary Scott
.n John B. Doe
.n Mary B. Smith
.n Mark E. Smith
```

Output 7.2 SAS Data Set with NAME Variable

OBS	NAME
1	Ann Lincoln
2	Mary Scott
3	John B. Doe
4	Mary B. Smith
5	Mark E. Smith

Note: if the system option CAPS is in effect, the job above never finds a CODE value of .n since SAS changes all characters in the job to uppercase before executing the job. See "SAS Options" for details.

DO WHILE You can create the same data set and external file with this DATA step using a DO WHILE statement:

```
DATA NEW;
   KEEP NAME;
   INFILE IN END=EOF;
   FILE OUT;
   INPUT CODE $2. @;
   DO WHILE(CODE='.n');
     PUT _INFILE_;
     INPUT NAME $CHAR25.;
     OUTPUT;
     IF EOF THEN STOP;
     INPUT CODE $2. @;
     END;
```

The first few statements in the step above are identical to those in the previous example except that here the KEEP statement appears first in the DATA step. SAS evaluates the expression in the DO WHILE statement at the beginning of the loop before the statements in the DO group are executed. When SAS reads a CODE value of .n:

1. PUT _INFILE_ writes the input buffer to file OUT.
2. The INPUT statement reads the NAME field from the current record.
3. The OUTPUT statement causes the observation to be written to SAS data set NEW.
4. The next INPUT statement brings in a new record and reads the CODE value.

If the CODE value just read is again .n, SAS repeats the statements just described. If not, SAS returns to the beginning of the DATA step and a new record.

Although not required, the STOP statement prevents a lost card message if SAS reaches end-of-file before all the INPUT statements in the step have been executed.

DO UNTIL The next DATA step uses a DO UNTIL statement to create the SAS data set and the output file:

```
DATA NEW(KEEP=NAME);
   INFILE IN END=EOF;
   FILE OUT;
   DO UNTIL(CODE='.n');
     IF EOF THEN STOP;
     INPUT CODE $2. NAME $CHAR25.;
     END;
   PUT _INFILE_;
```

The expression in a DO UNTIL statement is evaluated at the **bottom** of the loop after the statements in the DO loop are executed. Thus, you can include a variable in the expression that has not yet been given a value. (Note that the first occurrence of the variable in the DATA step determines the variable's attributes.)

The DATA step above reads records from INFILE IN until it finds a CODE value equal to .n. In this DATA step the trailing at sign is not used in the INPUT statement, since that would cause SAS to read from the same record, two bytes at

a time, looking for the string .n. When the CODE value just read is .n, SAS drops out of the DO loop, executes the PUT _INFILE_ statement, automatically outputs a record to SAS data set NEW, and returns to the beginning of the step for a new execution. If the DATA step included an INPUT statement before the DO UNTIL, the first record in INFILE IN would be lost, since a DO UNTIL is always executed at least one time.

The STOP statement again prevents a lost card message from occurring. A KEEP= data set option giving the variables to appear in the SAS data set is used in the example.

Another DO WHILE and DO UNTIL example Suppose your file of prospective clients contains codes only for the comment records. The first comment record for an individual or prospective client has a .c in the first two bytes of the record; the series of comment records ends with a record containing only a ## in the first two bytes. The data are shown below:

```
Ann Lincoln
101 S. Main St.
Anytown, USA
.c good prospect
called 01MAY84
called 04AUG84
##
Mary Scott
1 Castle Street
Scotland
.c referred by J. B. King; called 23APR85;
personal visit 01MAY85
##
John B. Doe
P.O. Box 3939
University of Anystate
Local, USA
Mary B. Smith
Mark E. Smith
123 Home Street
Hometown, USA
.c called 5MAY85
doesn't seem interested now
##
```

Your goal is to create an external file with only the names and addresses from the original file. The DATA steps below create the file without comment records. The new file is described in the control language by fileref OUT.

```
DATA _NULL_;
   INFILE IN;
   FILE OUT;
   INPUT REC $CHAR50.;
   IF REC=:'.c' THEN DO WHILE(REC¬='##');
      INPUT REC $CHAR50.;
      END;
   ELSE PUT REC;
```

The first INPUT statement reads the entire record at once and stores it in the variable REC.

If the value of REC begins with the characters .c, then SAS executes the DO WHILE statement. (See "SAS Expressions" for more about using the colon (:) in character comparisons.) Within the DO WHILE group, each time the current REC value is not ## SAS reads another record, replacing the current REC value in the program data vector with a new one. This process continues until the value of REC is ##. Then SAS exits from the DO WHILE loop and returns to the beginning of the DATA step and a new record.

If the value of REC does not begin with .c, SAS skips the DO WHILE group and writes the value of REC to the file with fileref OUT.

The lines written by the PUT statements above are shown below in **Output 7.3.**

Output 7.3 File with Name and Address Records

```
Ann Lincoln
101 S. Main St.
Anytown, USA
Mary Scott
1 Castle Street
Scotland
John B. Doe
P.O. Box 3939
University of Anystate
Local, USA
Mary B. Smith
Mark E. Smith
123 Home Street
Hometown, USA
```

You can replace the DO WHILE statement with a DO UNTIL in the example above, with the same results.

```
DATA _NULL_;
   INFILE IN;
   FILE OUT;
   INPUT REC $CHAR50.;
   IF REC=:'.c' THEN DO UNTIL(REC='##');
      INPUT REC $CHAR50.;
      END;
   ELSE PUT REC;
```

Jumping out of a loop Once SAS is executing a loop, it is possible to jump out of the loop at any point. You can jump out long enough to execute a series of SAS statements and then return. Or, you can end the loop permanently for that execution of the DATA step.

Output 7.4 shows a list of bank customers along with their initial investments in a savings account. The data are stored in a SAS data set named BANK. (The variable DATE is the date on which the initial investment was deposited. The variable is stored in the SAS data set as a SAS date value and printed using the DATE7. format.)

Output 7.4 SAS Data Set Containing Bank Records

OBS	ACCT_NO	AMT	DATE	PERCENT	TYPE
1	12321	1000	12OCT84	0.0550	A
2	24242	200	01JUL84	0.0560	M
3	54321	500	23JAN84	0.0525	A
4	89921	700	23APR84	0.0525	M

You want to calculate the interest earned for each customer, add it to the initial investment, and obtain a new balance. Interest is paid on the balance in savings accounts of type A on a certain day of the year. Type M savings accounts pay interest every 30 days. In the following program, run on the day of payment to A accounts, the balance in M accounts is calculated in an iterative DO loop; A accounts are handled by the statement following the label CALC.

```
DATA NEWBAL;
   SET BANK;
   INITIAL=AMT;
   TODAY=TODAY( );
   IF TYPE='M' THEN DO;
      MONTH=INT((TODAY-DATE)/30);
      PERCENT=PERCENT/12;
      DO I=1 TO MONTH;
         LINK CALC;
         END;
      RETURN;
      END;
CALC: AMT=AMT+AMT*PERCENT;
   RETURN;
```

The DATA step above illustrates several SAS features:

- nested DO loops
- jumping out of a DO loop and returning
- using an expression as the stopping value for a DO-group index variable
- datetime functions.

The resulting data set is NEWBAL, which is shown in **Output 7.5**.

Output 7.5 New Data Set after Interest Calculated

OBS	ACCT_NO	AMT	DATE	PERCENT	TYPE	INITIAL	TODAY	MONTH	I
1	12321	1055.00	12OCT84	0.0550000	A	1000	9147	.	.
2	24242	205.67	01JUL84	0.0046667	M	200	9147	6	7
3	54321	526.25	23JAN84	0.0525000	A	500	9147	.	.
4	89921	724.88	23APR84	0.0043750	M	700	9147	8	9

SAS reads observations from SAS data set BANK and stores the initial investment amounts (AMT) in variable INITIAL. TODAY is the SAS date value corresponding to the day the program is run. The number of full (30-day) months (MONTH) is used to determine the number of times the iterative DO loop is executed.

When the TYPE value is M, the percent of interest is divided by 12. Then, for each month SAS jumps to the statement labeled CALC and adds the interest to the AMT value.

When the TYPE value is not M, SAS skips the outside DO group and executes statements starting with the statement labeled CALC.

Jumping to the end of the DO group Sometimes you want to end the current execution of a DO loop before all the statements in the loop have been executed, and return to the top of the loop for another execution. You can jump to the END statement for the loop; SAS increments the index for the loop and continues processing.

For example, suppose you want to calculate anticipated monthly sales for each sales representative in a company for determining the annual budget and setting individual goals. The representative's name, length of service with the company, and average monthly sales for the preceding year are stored in a SAS data set named SALES, which is shown in **Output 7.6.**

Output 7.6 Sales Data

OBS	NAME	YEARS	AVERAGE
1	Jones	3	543
2	Smith	12	1620
3	Thomas	1	210
4	Marks	6	895
5	Adams	1	356
6	Doe	2	250

Each sales representative is expected to increase his or her average monthly sales from the preceding year by five percent in January, five and a half percent in February, six percent in March, six and a half percent in April, and so on for the year. Those employed for more than two years are entitled to a bonus if their total sales for the year reach $10,000. Those eligible for a bonus are output to another SAS data set. The DATA step below creates two SAS data sets: one containing anticipated sales for each sales representative for each month in the coming year, the other containing the names of sales representatives who earned bonuses:

```
DATA YEARLY(KEEP=NAME MONTH MONTHTOT) BONUS(KEEP=NAME TOTAL);
   SET SALES;
   TOTAL=0;
   INCREASE=.045;
   DO MONTH='JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
         'AUG', 'SEP', 'OCT', 'NOV', 'DEC';
      INCREASE=INCREASE+.005;
      MONTHTOT=AVERAGE+AVERAGE*INCREASE;
      OUTPUT YEARLY;
      TOTAL=TOTAL+MONTHTOT;
      IF YEARS<=2 THEN GO TO SKIP;
      IF MONTH='DEC' THEN IF TOTAL>=10000 THEN OUTPUT BONUS;
SKIP: END;
```

The DATA step above illustrates

- creating more than one SAS data set in a single DATA step
- keeping different subsets of variables on data sets created

- using a character variable to index a DO loop
- jumping to the END statement of a DO loop.

SAS first reads an observation from the SALES data set and initializes the variables TOTAL and INCREASE for that representative. Then the DO loop creates new observations for each value of MONTH.

After each observation is output to YEARLY, if the YEARS value is two or less, SAS skips to the END statement. This causes SAS to return to the top of the DO loop, increment the MONTH value, and generate another observation. The resulting data set YEARLY contains twelve observations for each observation from SALES. Each new observation contains a representative's expected sales for that month. See **Output 7.7**.

Output 7.7 Expected Sales per Month for Each Representative

DATA SET YEARLY

OBS	NAME	MONTH	MONTHTOT
1	Jones	JAN	570.15
2	Jones	FEB	572.86
3	Jones	MAR	575.58
4	Jones	APR	578.29
5	Jones	MAY	581.01
6	Jones	JUN	583.72
7	Jones	JUL	586.44
8	Jones	AUG	589.15
9	Jones	SEP	591.87
10	Jones	OCT	594.58
11	Jones	NOV	597.30
12	Jones	DEC	600.01
13	Smith	JAN	1701.00
14	Smith	FEB	1709.10
15	Smith	MAR	1717.20
16	Smith	APR	1725.30
17	Smith	MAY	1733.40
18	Smith	JUN	1741.50
19	Smith	JUL	1749.60
20	Smith	AUG	1757.70
21	Smith	SEP	1765.80
22	Smith	OCT	1773.90
23	Smith	NOV	1782.00
24	Smith	DEC	1790.10
25	Thomas	JAN	220.50
26	Thomas	FEB	221.55
27	Thomas	MAR	222.60
28	Thomas	APR	223.65
29	Thomas	MAY	224.70
30	Thomas	JUN	225.75
31	Thomas	JUL	226.80
32	Thomas	AUG	227.85
33	Thomas	SEP	228.90
34	Thomas	OCT	229.95
35	Thomas	NOV	231.00
36	Thomas	DEC	232.05
37	Marks	JAN	939.75
38	Marks	FEB	944.22
39	Marks	MAR	948.70
40	Marks	APR	953.17
41	Marks	MAY	957.65
42	Marks	JUN	962.12
43	Marks	JUL	966.60
44	Marks	AUG	971.07
45	Marks	SEP	975.55
46	Marks	OCT	980.02
47	Marks	NOV	984.50
48	Marks	DEC	988.97
49	Adams	JAN	373.80
50	Adams	FEB	375.58
51	Adams	MAR	377.36
52	Adams	APR	379.14
53	Adams	MAY	380.92
54	Adams	JUN	382.70

DATA SET YEARLY

OBS	NAME	MONTH	MONTHTOT
55	Adams	JUL	384.48
56	Adams	AUG	386.26
57	Adams	SEP	388.04
58	Adams	OCT	389.82
59	Adams	NOV	391.60
60	Adams	DEC	393.38
61	Doe	JAN	262.50
62	Doe	FEB	263.75
63	Doe	MAR	265.00
64	Doe	APR	266.25
65	Doe	MAY	267.50
66	Doe	JUN	268.75
67	Doe	JUL	270.00
68	Doe	AUG	271.25
69	Doe	SEP	272.50
70	Doe	OCT	273.75
71	Doe	NOV	275.00
72	Doe	DEC	276.25

When the value of YEARS is more than two, SAS checks to see if the TOTAL sales figure date exceeds $10,000. Observations whose total sales exceed that amount are output to SAS data set BONUS at the end of the DO loop. (The MONTH='DEC' condition ensures that an observation is output to BONUS only once.) See **Output 7.8**.

Output 7.8 Data Set Showing Bonuses

DATA SET BONUS

OBS	NAME	TOTAL
1	Smith	20946.6
2	Marks	11572.3

The observations in YEARLY provide the information necessary to determine anticipated sales for the coming year. For example, these statements use the SUMMARY procedure for descriptive statistics to summarize total sales for each month:

```
PROC SUMMARY DATA=YEARLY;
   CLASS MONTH;
   VAR MONTHTOT;
   OUTPUT OUT=SUMMARY SUM=TOTSALES;
```

The previous DATA step illustrates what happens when you jump to the END statement of a DO loop. You can create the same YEARLY and BONUS data sets in this simpler DATA step:

```
DATA YEARLY(KEEP=NAME MONTH MONTHTOT) BONUS(KEEP=NAME TOTAL);
   SET SALES;
   TOTAL=0;
   INCREASE=.045;
   DO MONTH='JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
         'AUG', 'SEP', 'OCT', 'NOV', 'DEC';
      INCREASE=INCREASE+.005;
      MONTHTOT=AVERAGE+AVERAGE*INCREASE;
      OUTPUT YEARLY;
      TOTAL+MONTHTOT;
```

```
   IF YEARS>2 THEN IF MONTH='DEC' THEN
      IF TOTAL>=10000 THEN OUTPUT BONUS;
   END;
```

Which data set would be printed if the statement

```
PROC PRINT;
```

had followed the DATA step above? When no data set name is specified in a PROC statement, SAS uses the most recently created data set. When more than one data set is created in a step, which is the most recently created?

The answer: when more than one data set is being created in a DATA step, the last data set created is the last one named in the DATA statement. Thus, data set BONUS would be printed by the PROC PRINT statement above.

Jumping out of a loop permanently Sometimes you want to jump completely out of a DO loop before the index variable has reached its stopping value. Perhaps you are indexing through the variables in an array and want to stop processing when you find the value for which you are searching.

For example, suppose you have a data set containing monthly gross salaries for each employee in a company. You want to deduct FICA tax from each salary value; tax is paid on every dollar earned up to $39,600.

In the statements below, the array SALARY contains twelve variables corresponding to the months of the year; the value of each variable is monthly gross income. You want to deduct FICA tax from each month's salary and store the adjusted salaries back in the SALARY array.

SAS data set EMPLOYEE contains the gross salaries of employees. See **Output 7.9**.

Output 7.9 Data Set Showing Monthly Salaries

EMPLOYEE

OBS	NAME	MONTH1	MONTH2	MONTH3	MONTH4	MONTH5	MONTH6	MONTH7	MONTH8	MONTH9	MONTH10	MONTH11	MONTH12
1	ADAMS	2550	2550	2550	2550	2550	2550	2600	2600	2600	2600	2600	2600
2	BROWN	2850	2850	2850	2850	2850	2850	2950	2950	2950	2950	2950	2950
3	CARTER	3850	3850	3850	3850	3850	3850	4000	4000	4000	4000	4000	4000
4	DAVIS	3550	3550	3550	3550	3550	3550	3700	3700	3700	3700	3700	3700
5	EDEN	4050	4050	4050	4050	4050	4050	4256	4256	4256	4256	4256	4256

```
DATA TOTAL;
   ARRAY SALARY(12) MONTH1-MONTH12;
   SET EMPLOYEE;
   MAX=0;
   DO I=1 TO 12;
      MAX=MAX+SALARY(I);
      IF MAX>=39600 THEN DO;
         SALARY(I)=SALARY(I)-(SALARY(I)-(MAX-39600))*.0705;
         GO TO OUT;
         END;
      ELSE SALARY(I)=SALARY(I)-(SALARY(I)*.0705);
      END;
OUT: ; or other SAS statements
```

This example illustrates

- an explicitly subscripted ARRAY statement
- processing an array in an iterative DO group
- jumping out of a DO group permanently
- a null statement.

The resulting data set is TOTAL, which is shown in **Output 7.10**.

Output 7.10 Salaries after FICA Tax

TOTAL

OBS	MONTH1	MONTH2	MONTH3	MONTH4	MONTH5	MONTH6	MONTH7	MONTH8	MONTH9	MONTH10	MONTH11	MONTH12	NAME	MAX	I
1	2370.22	2370.22	2370.22	2370.22	2370.22	2370.22	2416.70	2416.70	2416.70	2416.70	2416.70	2416.70	ADAMS	30900	13
2	2649.07	2649.07	2649.07	2649.07	2649.07	2649.07	2742.02	2742.02	2742.02	2742.02	2742.02	2742.02	BROWN	34800	13
3	3578.58	3578.58	3578.58	3578.58	3578.58	3578.58	3718.00	3718.00	3718.00	3718.00	3964.75	4000.00	CARTER	43100	11
4	3299.73	3299.73	3299.73	3299.73	3299.73	3299.73	3439.15	3439.15	3439.15	3439.15	3453.25	3700.00	DAVIS	39800	11
5	3764.48	3764.48	3764.48	3764.48	3764.48	3764.48	3955.95	3955.95	3955.95	4077.49	4256.00	4256.00	EDEN	41324	10

The iterative DO statement executes twelve times, once for each variable in the array. The index variable I is included in the new data set unless you drop it.

To process each observation from data set EMPLOYEE, SAS

- totals the unadjusted monthly salaries (MAX).
- deducts the remaining FICA tax when MAX reaches $39,600, then jumps out of the DO loop. Note that at this point no further salaries are added to MAX so its value in data set TOTAL is not the total salary for an employee.
- deducts the FICA tax from monthly salary and replaces the old salary with the adjusted salary when MAX has not reached the maximum.

To discontinue array processing in the example above, you jump out of the DO loop to a null statement and the end of the DATA step. Thus, a RETURN statement could replace the GO TO statement with the same result. However, you can continue processing observations after jumping out of the loop if other SAS statements appear after the label OUT.

More array examples Suppose you are coding responses to a questionnaire in which customers voted on selected enhancements to a product. They were each given 50 points to divide among twenty-five possible changes. A respondent could weight a few items more heavily by placing most of the 50 points on those items or could vote on more changes by spreading the 50 points over a larger number of items.

The item number and number of points are coded in sequence for each item voted on along with the respondent's ID. For example, the possible changes are numbered 1 to 25. The customer with ID 903 voted on four items:

- item 5: 10 points
- item 16: 10 points
- item 17: 10 points
- item 24: 20 points,

and has this data line:

```
903  5 10 16 10 17 10 24 20
```

The responses are stored in a file with fileref RESPOND. This DATA step reads the raw data and creates a SAS data set:

```
DATA RESULTS;
   INFILE RESPOND MISSOVER;
   ARRAY ALL{*} ITEM1-ITEM25;
   INPUT ID 1-3 @;
   DO UNTIL(I=. OR I=25);
      INPUT I @;
      IF I¬=. THEN INPUT ALL{I} @;
      END;
```

This example illustrates

- MISSOVER option of INFILE statement
- ARRAY with DO UNTIL
- complex expression in DO UNTIL
- reading the value of the DO UNTIL variable in an INPUT statement
- using an array reference in an INPUT statement.

See **Output 7.11**.

Output 7.11 Votes for Twenty-five Items

DATA SET RESULTS

OBS	ITEM1	ITEM2	ITEM3	ITEM4	ITEM5	ITEM6	ITEM7	ITEM8	ITEM9	ITEM10	ITEM11	ITEM12	ITEM13	ITEM14	ITEM15	ITEM16	ITEM17	ITEM18	ITEM19	ITEM20	ITEM21	ITEM22	ITEM23	ITEM24	ITEM25	ID	I
1	.	25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	25	.	.	.	.	901	.
2	.	.	10	.	.	.	.	.	.	.	.	.	.	15	.	.	.	.	.	.	.	.	25	.	.	902	.
3	.	.	.	.	10	.	.	.	.	.	.	.	.	.	.	10	10	.	.	.	.	.	.	20	.	903	.
4	2	2	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	45	904	25
5	.	.	.	.	.	.	.	.	.	.	.	.	50	.	.	.	.	.	.	.	.	.	.	.	.	905	.
6	.	.	.	.	.	.	.	.	.	.	.	45	.	.	.	.	.	.	.	.	.	.	.	.	.	906	.
7	.	.	.	.	.	.	.	.	.	40	5	5	.	.	.	.	.	.	.	.	.	.	.	.	.	907	.

Since a customer's response is always coded on one record, the MISSOVER option prevents SAS from going past the end of a record to read values. Instead, any values not found in the current record are set to missing in the SAS data set. Thus, when all responses have been read from an individual record, I is either a missing value (because of MISSOVER) or 25 (the last item in the questionnaire).

The INPUT statement inside the DO loop first reads a value for I. If I is not missing, I's value determines which element of the array is to be referenced in the INPUT statement by the array reference ALL{I}.

Multidimensional array processing Suppose the twenty-five questionnaire items in the example above relate to five different products. Besides the customer's ID, each item voted on is coded with three values: a product code (1 to 5), an item number (1 to 5), and the number of points voted.

The data are shown below. The first line represents user 801, who voted as follows:

- product 1, item 2: 25 points
- product 5, item 1: 25 points.

```
801 1 2 25 5 1 25
802 1 3 10 3 4 15 5 3 25
803 1 5 10 4 1 10 4 2 10 5 4 20
804 1 1 2 1 2 2 1 3 1 5 5 45
805 3 3 50
```

You can use multidimensional array processing to assign the data values to the correct variable ITEM1-ITEM25.[1] The resulting values are shown in **Output 7.12.**

Output 7.12 Values Assigned by Multidimensional Array Processing

RESULTS2

OBS	ITEM1	ITEM2	ITEM3	ITEM4	ITEM5	ITEM6	ITEM7	ITEM8	ITEM9	ITEM10	ITEM11	ITEM12	ITEM13	ITEM14	ITEM15	ITEM16	ITEM17	ITEM18	ITEM19	ITEM20	ITEM21	ITEM22	ITEM23	ITEM24	ITEM25	ID
1	.	25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	25	.	.	.	.	801
2	.	.	10	.	.	.	.	.	.	.	.	.	.	15	.	.	.	.	.	.	.	.	25	.	.	802
3	.	.	.	.	10	.	.	.	.	.	.	.	.	.	.	10	10	.	.	.	.	.	.	20	.	803
4	2	2	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	45	804
5	.	.	.	.	.	.	.	.	.	.	.	.	50	.	.	.	.	.	.	.	.	.	.	.	.	805

SAS Functions

The examples below illustrate the use of selected SAS functions.

Recoding using the PUT function Suppose you have data for a class of students and you want to print a report showing their average grades. Grades on each test range from 0 to 100. For the final grade, you want both a numeric grade and a letter grade (A,B,C,D, or F). Use PROC FORMAT to define letter grades for each range of numeric grades:

```
PROC FORMAT;
   VALUE GRADE 0-69='F'
      70-75='D'
      76-85='C'
      86-92='B'
      93-100='A';
```

You can associate this format with the variable containing the average grade using a PUT or FORMAT statement. However, if you want a new variable in the data set whose values are the letter grades, you can use a PUT function in a DATA step.

The data containing student grades are stored in a file with fileref CLASS:

```
NAME    TEST1    TEST2    TEST3    EXAM
ANN      84       87       82       92
BILL     E        75       79       79
CAROL    95       97      100       98
TOM      66       74       79       83
MARY     76       88       90       84
```

The letter E is coded for Bill's TEST1 score since he was excused on the day the test was given. The value is to be treated as a missing value in the calculation

of Bill's average grade. The DATA step below reads the data, averages the grades, and creates the two new variables AVERAGE and LETTER:

```
DATA FINAL;
   INFILE CLASS;
   MISSING E;
   INPUT NAME $ TEST1-TEST3 EXAM;
   AVERAGE=MEAN(OF TEST1-TEST3,EXAM);
   LETTER=PUT(AVERAGE,GRADE.);
```

This example illustrates

- MISSING statement
- special missing values
- MEAN function
- PUT function with user-defined format
- how SAS prints formats that do not fit.

The resulting data set is shown in **Output 7.13**.

Output 7.13 Put Function with Format Created by PROC FORMAT

OBS	NAME	TEST1	TEST2	TEST3	EXAM	AVERAGE	LETTER
1	ANN	84	87	82	92	86.2500	B
2	BILL	E	75	79	79	77.6667	C
3	CAROL	95	97	100	98	97.5000	A
4	TOM	66	74	79	83	75.5000	*
5	MARY	76	88	90	84	84.5000	C

Data set FINAL contains two calculated variables: AVERAGE, a numeric variable containing the mean of the test grades and exam grade, and LETTER, a character variable whose values are the letter grades corresponding to the average grades.

The MISSING statement tells SAS that the letter E, appearing in a numeric field, is not to be considered an invalid value but a special missing value. Since the MEAN function uses only nonmissing values in its calculations, Bill's missing TEST1 value is not used to calculate his average grade.

Note that in the resulting data set FINAL, the LETTER value for Tom is an asterisk (*). This value is returned since Tom's average grade falls out of the range specified in PROC FORMAT. Normally when a value is undefined by PROC FORMAT, SAS uses the original value. In this case, however, the value 75.5 is too large to fit into the one-character length of LETTER, so SAS prints an asterisk instead.

You can get around the problem by including the FUZZ= option in the VALUE statement of PROC FORMAT. The following VALUE statement could replace the earlier one:

```
VALUE GRADE (FUZZ=.5)
   0-69='F'
   70-75='D'
   76-85='C'
   86-92='B'
   93-100='A';
```

The VALUE statement causes the grade of 75.5 for TOM to be rounded to the category with label C. See **Output 7.14**.

Output 7.14 PROC FORMAT with FUZZ= Option

OBS	NAME	TEST1	TEST2	TEST3	EXAM	AVERAGE	LETTER
1	ANN	84	87	82	92	86.2500	B
2	BILL	E	75	79	79	77.6667	C
3	CAROL	95	97	100	98	97.5000	A
4	TOM	66	74	79	83	75.5000	C
5	MARY	76	88	90	84	84.5000	C

Note that if you are interested only in the variable LETTER, and not in its numeric counterpart AVERAGE, you can combine the last two statements in the DATA step above:

```
DATA FINAL;
   INFILE CLASS;
   MISSING E;
   INPUT NAME $ TEST1-TEST3 EXAM;
   LETTER=PUT(MEAN(OF TEST1-TEST3,EXAM),GRADE.);
```

Character to numeric: the INPUT function Suppose you have a character variable containing product codes for drugstore items. The product codes are thirteen characters long; the first three digits are an alphanumeric code describing the department that sells the product, the next three digits identify the product, and the last seven digits contain the date the product was purchased for resale. The variable's name is CODE in a SAS data set named PRODUCT.

Below are some typical CODE values. The first two from the pharmacy department identify products 001 and 201, which were purchased March 10 and April 29, 1985, respectively. The third item is from households; the fourth item is from the book department.

```
PHA00110MAR85
PHA20129APR85
HOU09918JAN84
BOO40402FEB85
```

You want to include the date part of the product code as a SAS date value so that you can determine when a product's shelf life has expired. The DATA step below creates a numeric variable named DATE containing the SAS date value corresponding to the date part of the CODE values:

```
DATA PRODUCT2;
   SET PRODUCT;
   DATE=INPUT(SUBSTR(CODE,7,7),DATE7.);
```

This DATA step illustrates

- using SAS functions to generate arguments to functions
- SUBSTR function
- INPUT function
- DATE7. informat.

The resulting data set PRODUCT2 is shown in **Output 7.15**.

Output 7.15 Result of SUBSTR and INPUT Functions

PRODUCT2

OBS	CODE	DATE
1	PHA00110MAR85	9200
2	PHA20129APR85	9250
3	HOU09918JAN84	8783
4	BOO40402FEB85	9164

The SUBSTR function causes only the last seven characters of CODE to be read by the INPUT function using the DATE7. informat. Note that the value of DATE is stored in the new data set as the number of days between the date read and January 1, 1960.

A FORMAT statement added to the DATA step associates a date format with the SAS date value. For example, if the following FORMAT statement is added to the DATA step above:

```
FORMAT DATE WORDDATE.;
```

any SAS procedure uses the associated format WORDDATE. to print values of DATE:

```
PROC PRINT DATA=PRODUCT2;
```

See **Output 7.16**.

Output 7.16 Formatted SAS Date Values

PRODUCT2 WITH FORMATTED DATE VALUES

OBS	CODE	DATE
1	PHA00110MAR85	MARCH 10, 1985
2	PHA20129APR85	APRIL 29, 1985
3	HOU09918JAN84	JANUARY 18, 1984
4	BOO40402FEB85	FEBRUARY 2, 1985

The contents of the data set show that the DATE variable is numeric and has the format WORDDATE associated with it:

```
PROC CONTENTS DATA=PRODUCT2;
```

See **Output 7.17**.

Output 7.17 PROC CONTENTS Showing Variable's Type and Format

```
                                    CONTENTS PROCEDURE


                                CONTENTS OF SAS MEMBER WORK.PRODUCT2


CREATED BY OS JOB PRODUCT   ON CPUID XX-XXXX-XXXXXX   AT 14:16 SUNDAY, JANUARY 13, 1985   BY SAS RELEASE 5.XX
DSNAME=XXXXXXXX.XXXXXXX.XXXXX.XXXXXXX.XXXXXXXX   OBSERVATIONS PER TRACK =762   BLKSIZE=19054   LRECL=25
GENERATED BY DATA
NUMBER OF OBSERVATIONS: 4   NUMBER OF VARIABLES: 2
MEMTYPE:   DATA

                            ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  CODE      CHAR     13        4
  2  DATE      NUM       8       17  WORDDATE18.


                  ------------------------------ SOURCE RECORDS ------------------------------
                  |    DATA PRODUCT2;                                                        |
                  |       SET PRODUCT;                                                       |
                  |       DATE=INPUT(SUBSTR(CODE,7,7),DATE7.);                               |
                  |       FORMAT DATE WORDDATE.;                                             |
                  |                                                                          |
                  ----------------------------------------------------------------------------
```

WORKING WITH SAS DATA SETS: SET, MERGE, AND UPDATE

This section of the chapter describes how to work with existing SAS data sets using the SET, MERGE, and UPDATE statements. Additional examples using these statements are given with the statement descriptions and in the *SAS Applications Guide*.

Comparison of Methods: Combining SAS Data Sets

Suppose you have collected sales data for the first two months of the year. Each month's data are stored in a separate SAS data set, and each data set contains the variables REGION, SALESREP, and SALESAMT with corresponding values for the month. See **Output 7.18**.

Output 7.18 Input Data Sets

JAN

OBS	REGION	SALESREP	SALESAMT
1	EAST	STAFER	9664
2	EAST	YOUNG	22969
3	EAST	STRIDE	27253
4	WEST	VETTER	38928
5	WEST	CURCI	21531
6	WEST	GRECO	18523
7	WEST	RYAN	32915
8	WEST	TOMAS	42109
9	SOUTH	ROCKFELT	38737
10	SOUTH	MOORE	25718
11	SOUTH	STELAM	27926
12	NORTH	FARLOW	32719
13	NORTH	SMITH, T	38712

FEB

OBS	REGION	SALESREP	SALESAMT
1	EAST	YOUNG	20866
2	EAST	STRIDE	29100
3	EAST	ISSAC	21991
4	WEST	CURCI	18432
5	WEST	GRECO	21497
6	WEST	RYAN	33041
7	WEST	TOMAS	38300
8	SOUTH	ROCKFELT	34212
9	SOUTH	MOORE	23312
10	SOUTH	STELAM	25371
11	NORTH	FARLOW	29219
12	NORTH	SMITH, T	31909
13	NORTH	RICHARDS	11927

These data sets can be combined in several ways: concatenating, interleaving, or merging.

Concatenating You can create a data set with observations for each sales representative for each month by using the concatenation operation. In this case you want to create a new variable containing the current value for the current month:

```
DATA CONCAT;
   SET JAN(IN=J) FEB(IN=F);
   IF J THEN MONTH='JAN';
   ELSE MONTH='FEB';
```

The special IN= variables are used with each data set named in the SET statement to indicate the data set from which SAS read the current observation.

The resulting data set CONCAT contains as many observations as there are in the total of JAN and FEB. See **Output 7.19**.

Output 7.19 Result of Concatenation

CONCAT

OBS	REGION	SALESREP	SALESAMT	MONTH
1	EAST	STAFER	9664	JAN
2	EAST	YOUNG	22969	JAN
3	EAST	STRIDE	27253	JAN
4	WEST	VETTER	38928	JAN
5	WEST	CURCI	21531	JAN
6	WEST	GRECO	18523	JAN
7	WEST	RYAN	32915	JAN
8	WEST	TOMAS	42109	JAN
9	SOUTH	ROCKFELT	38737	JAN
10	SOUTH	MOORE	25718	JAN
11	SOUTH	STELAM	27926	JAN
12	NORTH	FARLOW	32719	JAN
13	NORTH	SMITH, T	38712	JAN
14	EAST	YOUNG	20866	FEB
15	EAST	STRIDE	29100	FEB
16	EAST	ISSAC	21991	FEB
17	WEST	CURCI	18432	FEB
18	WEST	GRECO	21497	FEB
19	WEST	RYAN	33041	FEB
20	WEST	TOMAS	38300	FEB
21	SOUTH	ROCKFELT	34212	FEB
22	SOUTH	MOORE	23312	FEB
23	SOUTH	STELAM	25371	FEB
24	NORTH	FARLOW	29219	FEB
25	NORTH	SMITH, T	31909	FEB
26	NORTH	RICHARDS	11927	FEB

CONCAT observations appear in the order they were read from the two monthly data sets: all observations from JAN followed by all observations from FEB.

The concatenation of SAS data sets is also performed by the SAS procedure APPEND, which copies observations from one SAS data set to the end of another. See the APPEND procedure description for more information.

Interleaving If you want your new data set to include all the observations from the JAN and FEB data sets in order of REGION, you can interleave the two data sets. An interleave operation uses a BY statement in the DATA step; the interleaved data sets must be sorted in the order described by the BY statement.

```
PROC SORT DATA=JAN;
   BY REGION;
PROC SORT DATA=FEB;
   BY REGION;
DATA INTERLV;
   SET JAN(IN=J) FEB(IN=F);
   BY REGION;
   IF J THEN MONTH='JAN';
   ELSE MONTH='FEB';
```

Observations in SAS data set INTERLV are sorted by REGION. For each region, all observations from JAN are followed by all FEB observations. See **Output 7.20**.

Output 7.20 Result of Interleaving

INTERLV

OBS	REGION	SALESREP	SALESAMT	MONTH
1	EAST	STAFER	9664	JAN
2	EAST	YOUNG	22969	JAN
3	EAST	STRIDE	27253	JAN
4	EAST	YOUNG	20866	FEB
5	EAST	STRIDE	29100	FEB
6	EAST	ISSAC	21991	FEB
7	NORTH	FARLOW	32719	JAN
8	NORTH	SMITH, T	38712	JAN
9	NORTH	FARLOW	29219	FEB
10	NORTH	SMITH, T	31909	FEB
11	NORTH	RICHARDS	11927	FEB
12	SOUTH	ROCKFELT	38737	JAN
13	SOUTH	MOORE	25718	JAN
14	SOUTH	STELAM	27926	JAN
15	SOUTH	ROCKFELT	34212	FEB
16	SOUTH	MOORE	23312	FEB
17	SOUTH	STELAM	25371	FEB
18	WEST	VETTER	38928	JAN
19	WEST	CURCI	21531	JAN
20	WEST	GRECO	18523	JAN
21	WEST	RYAN	32915	JAN
22	WEST	TOMAS	42109	JAN
23	WEST	CURCI	18432	FEB
24	WEST	GRECO	21497	FEB
25	WEST	RYAN	33041	FEB
26	WEST	TOMAS	38300	FEB

If you want your new data set to have observations for each SALESREP appearing together, use these statements:

```
PROC SORT DATA=JAN;
   BY REGION SALESREP;
PROC SORT DATA=FEB;
   BY REGION SALESREP;
```

```
DATA INTERLV2;
   SET JAN(IN=J) FEB(IN=F);
   BY REGION SALESREP;
   IF J THEN MONTH='JAN';
   ELSE MONTH='FEB';
```

The result is shown in **Output 7.21.**

Output 7.21 Result of Interleaving with Two BY Variables

INTERLV2

OBS	REGION	SALESREP	SALESAMT	MONTH
1	EAST	ISSAC	21991	FEB
2	EAST	STAFER	9664	JAN
3	EAST	STRIDE	27253	JAN
4	EAST	STRIDE	29100	FEB
5	EAST	YOUNG	22969	JAN
6	EAST	YOUNG	20866	FEB
7	NORTH	FARLOW	32719	JAN
8	NORTH	FARLOW	29219	FEB
9	NORTH	RICHARDS	11927	FEB
10	NORTH	SMITH, T	38712	JAN
11	NORTH	SMITH, T	31909	FEB
12	SOUTH	MOORE	25718	JAN
13	SOUTH	MOORE	23312	FEB
14	SOUTH	ROCKFELT	38737	JAN
15	SOUTH	ROCKFELT	34212	FEB
16	SOUTH	STELAM	27926	JAN
17	SOUTH	STELAM	25371	FEB
18	WEST	CURCI	21531	JAN
19	WEST	CURCI	18432	FEB
20	WEST	GRECO	18523	JAN
21	WEST	GRECO	21497	FEB
22	WEST	RYAN	32915	JAN
23	WEST	RYAN	33041	FEB
24	WEST	TOMAS	42109	JAN
25	WEST	TOMAS	38300	FEB
26	WEST	VETTER	38928	JAN

Match-merging Suppose that you want your combined data set to include one observation for each sales representative, with each monthly sales total included as a separate variable in that observation. Use the merge operation, matching on SALESREP.

```
PROC SORT DATA=JAN;
   BY SALESREP;
PROC SORT DATA=FEB;
   BY SALESREP;
DATA MATCH1;
   MERGE JAN(RENAME=(SALESAMT=JANSALES))
      FEB(RENAME=(SALESAMT=FEBSALES));
   BY SALESREP;
```

The MERGE operation, when combining observations, overlays variables with the same name from the data sets being merged. The result in the new data set is one variable of that name; its value is the value from the last data set appearing in the MERGE statement. Thus, in the example above, if the RENAME data set option is not used to change the name of the variable SALES, the January sales amount will be lost.

Data set MATCH1 is shown in **Output 7.22.**

Output 7.22 Result of Match-Merge

MATCH1

OBS	REGION	SALESREP	JANSALES	FEBSALES
1	WEST	CURCI	21531	18432
2	NORTH	FARLOW	32719	29219
3	WEST	GRECO	18523	21497
4	EAST	ISSAC	.	21991
5	SOUTH	MOORE	25718	23312
6	NORTH	RICHARDS	.	11927
7	SOUTH	ROCKFELT	38737	34212
8	WEST	RYAN	32915	33041
9	NORTH	SMITH, T	38712	31909
10	EAST	STAFER	9664	.
11	SOUTH	STELAM	27926	25371
12	EAST	STRIDE	27253	29100
13	WEST	TOMAS	42109	38300
14	WEST	VETTER	38928	.
15	EAST	YOUNG	22969	20866

Note that when a sales representative appears in only one of the data sets JAN or FEB, his sales value for the month he was not working is missing in the new data set.

In a match-merge, the matching variable should uniquely identify observations in the data sets being merged. Suppose that SALESREP values are not unique but that you need both REGION and SALESREP to identify each sales representative uniquely. Then you can sort and merge by these two variables. To get a match, both the REGION and then SALESREP values must be the same.

```
PROC SORT DATA=JAN;
   BY REGION SALESREP;
PROC SORT DATA=FEB;
   BY REGION SALESREP;
DATA MATCH2;
   MERGE JAN(RENAME=(SALESAMT=JANSALES))
      FEB(RENAME=(SALESAMT=FEBSALES));
   BY REGION SALESREP;
```

The resulting data set is in order of REGION and SALESREP. See **Output 7.23**.

Output 7.23 Result of Match-Merge with Two BY Variables

MATCH2

OBS	REGION	SALESREP	JANSALES	FEBSALES
1	EAST	ISSAC	.	21991
2	EAST	STAFER	9664	.
3	EAST	STRIDE	27253	29100
4	EAST	YOUNG	22969	20866
5	NORTH	FARLOW	32719	29219
6	NORTH	RICHARDS	.	11927
7	NORTH	SMITH, T	38712	31909
8	SOUTH	MOORE	25718	23312
9	SOUTH	ROCKFELT	38737	34212
10	SOUTH	STELAM	27926	25371
11	WEST	CURCI	21531	18432
12	WEST	GRECO	18523	21497
13	WEST	RYAN	32915	33041
14	WEST	TOMAS	42109	38300
15	WEST	VETTER	38928	.

Updating The UPDATE operation in SAS is similar to MERGE but is specialized to handle the function of updating a master file with transactions. In the example above, you do not want to lose the January sales amount from the combined data set, so the traditional update is inappropriate for creating a combined data set.

Instead, suppose that you have a master data set containing all sales representatives along with some background information about each of them. At the end of the month, you want to update this master with the monthly sales record for each representative.

SAS data set MASTER is shown in **Output 7.24**.

Output 7.24 Master Data Set

DATA SET MASTER

OBS	REGION	SALESREP	NAME	BEGDATE	SALES
1	EAST	GREEN	GREEN, ROBERT C.	02JAN84	551165
2	EAST	ISSAC	ISSAC, HOWARD T.	16JAN84	281910
3	EAST	STAFER	STAFER, MICHAEL C.	30JAN84	15458
4	EAST	STRIDE	STRIDE, JOHN M.	05APR84	498108
5	EAST	YOUNG	YOUNG, MARTIN V.	19APR84	354962
6	NORTH	FARLOW	FARLOW, DOUGLAS C.	08JUL85	447859
7	NORTH	RICHARDS	RICHARDS, BRENDA K.	22JUL85	169780
8	NORTH	SMITH, M	SMITH, MARY A.	11OCT85	508701
9	NORTH	SMITH, T	SMITH, THOMAS M.	25OCT85	748036
10	SOUTH	MOORE	MOORE, PATRICIA P.	30JAN85	480404
11	SOUTH	ROCKFELT	ROCKFELT, STEPHEN L.	05APR85	160566
12	SOUTH	STELAM	STELAM, MICHAEL O.	19APR85	644936
13	WEST	CURCI	CURCI, CYNTHIA L.	08JUL84	869282
14	WEST	GRECO	GRECO, ANTHONY T.	22JUL84	122341
15	WEST	HARRELL	HARRELL, DAVID R.	11OCT84	297837
16	WEST	RYAN	RYAN, HENRY H.	25OCT84	816871
17	WEST	TOMAS	TOMAS, JAMES M.	02JAN85	162072
18	WEST	VETTER	VETTER, WILLIAM A.	16JAN85	958498

The SALES variable in the MASTER data set contains the total-to-date sales for each representative. Every other month you want to update this total with the monthly sales (SALESAMT) from the previous two months.

This is not an update in the sense that you do not want to replace values in the MASTER with new values, but you can use programming statements to add each month's total to the value of SALES. The resulting data set is a copy of MASTER, but with new total SALES.

Any of the combined data sets created above could be used as the transaction data set and matched with the MASTER data set in the UPDATE operation. The three SAS data sets CONCAT, INTERLV, and INTERLV2 all contain the same observations, but in a different order.

The MASTER data set is sorted by REGION and alphabetically by SALESREP within REGION. Since the data set INTERLV2 is already sorted in that order, use INTERLV2 as a transaction data set with UPDATE.

```
DATA UPDATE;
   DROP MONTH SALESAMT;
   UPDATE MASTER INTERLV2;
   BY REGION SALESREP;
   SALES+SALESAMT;
```

The resulting data set UPDATE now includes the new total SALES. See **Output 7.25**.

Output 7.25 Data Set with Updated Values

UPDATE

OBS	REGION	SALESREP	NAME	BEGDATE	SALES
1	EAST	GREEN	GREEN, ROBERT C.	02JAN84	551165
2	EAST	ISSAC	ISSAC, HOWARD T.	16JAN84	303901
3	EAST	STAFER	STAFER, MICHAEL C.	30JAN84	25122
4	EAST	STRIDE	STRIDE, JOHN M.	05APR84	554461
5	EAST	YOUNG	YOUNG, MARTIN V.	19APR84	398797
6	NORTH	FARLOW	FARLOW, DOUGLAS C.	08JUL85	509797
7	NORTH	RICHARDS	RICHARDS, BRENDA K.	22JUL85	181707
8	NORTH	SMITH, M	SMITH, MARY A.	11OCT85	508701
9	NORTH	SMITH, T	SMITH, THOMAS M.	25OCT85	818657
10	SOUTH	MOORE	MOORE, PATRICIA P.	30JAN85	529434
11	SOUTH	ROCKFELT	ROCKFELT, STEPHEN L.	05APR85	233515
12	SOUTH	STELAM	STELAM, MICHAEL O.	19APR85	698233
13	WEST	CURCI	CURCI, CYNTHIA L.	08JUL84	909245
14	WEST	GRECO	GRECO, ANTHONY T.	22JUL84	162361
15	WEST	HARRELL	HARRELL, DAVID R.	11OCT84	297837
16	WEST	RYAN	RYAN, HENRY H.	25OCT84	882827
17	WEST	TOMAS	TOMAS, JAMES M.	02JAN85	242481
18	WEST	VETTER	VETTER, WILLIAM A.	16JAN85	997426

Notice that unlike MERGE, UPDATE outputs an observation only at the end of a BY group—in this case, only once for each unique combination of REGION and SALESREP. Thus, even though many sales representatives occur twice in the INTERLV2 data set (for example, STRIDE in region EAST), the SALESAMT values are added from both INTERLV2 observations before the new observation is output.

Note that in any combining operation, the resulting data set contains all the variables from the data sets being joined unless a DROP or KEEP statement or data set option subsets the variables.

If you always want the current master data set to have the name MASTER and the previous master to have the name UPDATE, use the DATASETS procedure to exchange the two names after performing the update:

```
PROC DATASETS LIBRARY=WORK;
   EXCHANGE MASTER=UPDATE;
```

(This example uses data sets in the WORK library; however, most applications would use permanent data sets.)

Suppose that you want to use the data set MATCH2, which resulted from the second merge example above, as your transaction data set in the update. MATCH2 has only one observation for every REGION/SALESREP combination. To perform the update, these statements are needed:

```
DATA UPDATE2;
   UPDATE MASTER MATCH2;
   BY REGION SALESREP;
   SALES+SUM(JANSALES,FEBSALES);
   DROP JANSALES FEBSALES;
```

The result is shown in **Output 7.26**.

Output 7.26 Second Data Set with Updated Values

UPDATE2

OBS	REGION	SALESREP	NAME	BEGDATE	SALES
1	EAST	GREEN	GREEN, ROBERT C.	02JAN84	551165
2	EAST	ISSAC	ISSAC, HOWARD T.	16JAN84	303901
3	EAST	STAFER	STAFER, MICHAEL C.	30JAN84	25122
4	EAST	STRIDE	STRIDE, JOHN M.	05APR84	554461
5	EAST	YOUNG	YOUNG, MARTIN V.	19APR84	398797
6	NORTH	FARLOW	FARLOW, DOUGLAS C.	08JUL85	509797
7	NORTH	RICHARDS	RICHARDS, BRENDA K.	22JUL85	181707
8	NORTH	SMITH, M	SMITH, MARY A.	11OCT85	508701
9	NORTH	SMITH, T	SMITH, THOMAS M.	25OCT85	818657
10	SOUTH	MOORE	MOORE, PATRICIA P.	30JAN85	529434
11	SOUTH	ROCKFELT	ROCKFELT, STEPHEN L.	05APR85	233515
12	SOUTH	STELAM	STELAM, MICHAEL O.	19APR85	698233
13	WEST	CURCI	CURCI, CYNTHIA L.	08JUL84	909245
14	WEST	GRECO	GRECO, ANTHONY T.	22JUL84	162361
15	WEST	HARRELL	HARRELL, DAVID R.	11OCT84	297837
16	WEST	RYAN	RYAN, HENRY H.	25OCT84	882827
17	WEST	TOMAS	TOMAS, JAMES M.	02JAN85	242481
18	WEST	VETTER	VETTER, WILLIAM A.	16JAN85	997426

Another example of combining SAS data sets in a way similar to MERGE and UPDATE is shown below in the section **Multiple SET Statements: Direct Access.**

More Than Two SAS Data Sets

The concatenation, interleave, and merge operations discussed above can be applied to as many as fifty SAS data sets in a single step. Suppose that the MERGE operation in the sales example is done quarterly and that you have collected sales data for March in a SAS data set named MAR, which is shown in **Output 7.27.**

Output 7.27 Input Data Set

MAR

OBS	REGION	SALESREP	SALESAMT
1	EAST	GREEN	24960
2	EAST	YOUNG	19866
3	EAST	STRIDE	24700
4	EAST	ISSAC	27997
5	WEST	CURCI	18432
6	WEST	GRECO	20497
7	WEST	RYAN	29047
8	WEST	TOMAS	30300
9	SOUTH	ROCKFELT	29272
10	SOUTH	MOORE	28372
11	NORTH	FARLOW	21279
12	NORTH	SMITH, T	30909
13	NORTH	RICHARDS	17927

Concatenating three data sets For example, to concatenate the three data sets JAN, FEB, and MAR, enter the statements

```
DATA QUARTER1;
   SET JAN(IN=J) FEB(IN=F) MAR(IN=M);
   IF J THEN MONTH='JAN';
   ELSE IF F THEN MONTH='FEB';
   ELSE MONTH='MAR';
```

The result is shown in **Output 7.28.**

Output 7.28 Result of Concatenating Three Data Sets

QUARTER1

OBS	REGION	SALESREP	SALESAMT	MONTH
1	EAST	STAFER	9664	JAN
2	EAST	YOUNG	22969	JAN
3	EAST	STRIDE	27253	JAN
4	WEST	VETTER	38928	JAN
5	WEST	CURCI	21531	JAN
6	WEST	GRECO	18523	JAN
7	WEST	RYAN	32915	JAN
8	WEST	TOMAS	42109	JAN
9	SOUTH	ROCKFELT	38737	JAN
10	SOUTH	MOORE	25718	JAN
11	SOUTH	STELAM	27926	JAN
12	NORTH	FARLOW	32719	JAN
13	NORTH	SMITH, T	38712	JAN
14	EAST	YOUNG	20866	FEB
15	EAST	STRIDE	29100	FEB
16	EAST	ISSAC	21991	FEB
17	WEST	CURCI	18432	FEB
18	WEST	GRECO	21497	FEB
19	WEST	RYAN	33041	FEB
20	WEST	TOMAS	38300	FEB
21	SOUTH	ROCKFELT	34212	FEB
22	SOUTH	MOORE	23312	FEB
23	SOUTH	STELAM	25371	FEB
24	NORTH	FARLOW	29219	FEB
25	NORTH	SMITH, T	31909	FEB
26	NORTH	RICHARDS	11927	FEB
27	EAST	GREEN	24960	MAR
28	EAST	YOUNG	19866	MAR
29	EAST	STRIDE	24700	MAR
30	EAST	ISSAC	27997	MAR
31	WEST	CURCI	18432	MAR
32	WEST	GRECO	20497	MAR
33	WEST	RYAN	29047	MAR
34	WEST	TOMAS	30300	MAR
35	SOUTH	ROCKFELT	29272	MAR
36	SOUTH	MOORE	28372	MAR
37	NORTH	FARLOW	21279	MAR
38	NORTH	SMITH, T	30909	MAR
39	NORTH	RICHARDS	17927	MAR

As before, SAS reads observations from one data set until the data set contains no more observations, then goes on to the next data set listed in the SET statement, and so on until it has read all the observations in all the data sets listed in the SET statement.

Match-merging three data sets The DATA step below merges the three SAS data sets JAN, FEB, and MAR, renaming the SALESAMT variable to correspond to the month to which the sales total refers.

```
PROC SORT DATA=JAN;
   BY REGION SALESREP;
PROC SORT DATA=FEB;
   BY REGION SALESREP;
PROC SORT DATA=MAR;
   BY REGION SALESREP;
DATA MATCH3;
   MERGE JAN(RENAME=(SALESAMT=JANSALES))
      FEB(RENAME=(SALESAMT=FEBSALES))
      MAR(RENAME=(SALESAMT=MARSALES));
   BY REGION SALESREP;
```

The resulting data set MATCH3 is shown in **Output 7.29**.

Output 7.29 Result of Match-Merging Three Data Sets

MATCH3

OBS	REGION	SALESREP	JANSALES	FEBSALES	MARSALES
1	EAST	GREEN	.	.	24960
2	EAST	ISSAC	.	21991	27997
3	EAST	STAFER	9664	.	.
4	EAST	STRIDE	27253	29100	24700
5	EAST	YOUNG	22969	20866	19866
6	NORTH	FARLOW	32719	29219	21279
7	NORTH	RICHARDS	.	11927	17927
8	NORTH	SMITH, T	38712	31909	30909
9	SOUTH	MOORE	25718	23312	28372
10	SOUTH	ROCKFELT	38737	34212	29272
11	SOUTH	STELAM	27926	25371	.
12	WEST	CURCI	21531	18432	18432
13	WEST	GRECO	18523	21497	20497
14	WEST	RYAN	32915	33041	29047
15	WEST	TOMAS	42109	38300	30300
16	WEST	VETTER	38928	.	.

You can merge up to fifty SAS data sets in a single DATA step. SAS keeps a pointer on the next observation in each data set being merged. It selects the observation with the lowest BY value, then brings in information from each data set containing that observation, reading from data sets in the order they appear in the MERGE statement. If a data set does not contain the current lowest BY value, variables appearing only in that data set have a missing value in the new, combined observation. Thus, observations like GREEN from the EAST, which appears only in the MAR data set, have missing values in MATCH3 for both the JAN and FEB variables.

Merging a few observations with all observations in a SAS data set: two SET statements You want to include in each observation in MATCH3 the overall average monthly sales for the company. With this value added to each observation, you can then calculate for each sales representative whether his or her sales are above or below the company average.

You can use the MEANS procedure to create an observation containing the average monthly sales to date. (Analyze the data set QUARTER1 created by contatenating the JAN, FEB, and MAR data sets.)

```
PROC MEANS DATA=QUARTER1;
   VAR SALESAMT;
   OUTPUT OUT=MEANOUT MEAN=AVERAGE;
```

The resulting observation, output from the MEANS procedure, is shown in **Output 7.30**.

Output 7.30 Data Set Created by PROC MEANS

MEANOUT

OBS	AVERAGE
1	26677.9

You can merge this observation containing one variable, AVERAGE, with each observation in data set MATCH3. Use these statements:

```
DATA COMBINE;
   IF _N_=1 THEN SET MEANOUT;
   SET MATCH3;
```

Data set COMBINE contains all the observations from MATCH3. Each observation is combined with the one observation in MEANOUT. See **Output 7.31.**

Output 7.31 Combining One Observation with All Observations in a SAS Data Set

COMBINE

OBS	AVERAGE	REGION	SALESREP	JANSALES	FEBSALES	MARSALES
1	26677.9	EAST	GREEN	.	.	24960
2	26677.9	EAST	ISSAC	.	21991	27997
3	26677.9	EAST	STAFER	9664	.	.
4	26677.9	EAST	STRIDE	27253	29100	24700
5	26677.9	EAST	YOUNG	22969	20866	19866
6	26677.9	NORTH	FARLOW	32719	29219	21279
7	26677.9	NORTH	RICHARDS	.	11927	17927
8	26677.9	NORTH	SMITH, T	38712	31909	30909
9	26677.9	SOUTH	MOORE	25718	23312	28372
10	26677.9	SOUTH	ROCKFELT	38737	34212	29272
11	26677.9	SOUTH	STELAM	27926	25371	.
12	26677.9	WEST	CURCI	21531	18432	18432
13	26677.9	WEST	GRECO	18523	21497	20497
14	26677.9	WEST	RYAN	32915	33041	29047
15	26677.9	WEST	TOMAS	42109	38300	30300
16	26677.9	WEST	VETTER	38928	.	.

Combining subtotals with each detail record Suppose that you have calculated averages for each REGION and want to combine the regional average sales with observations from that region. You can create observations containing the regional sales average using the MEANS procedure.

```
PROC SORT DATA=QUARTER1;
   BY REGION;
PROC MEANS DATA=QUARTER1;
   VAR SALESAMT;
   BY REGION;
   OUTPUT OUT=AVERAGE MEAN=AVGSALES;
```

Data set AVERAGE is shown in **Output 7.32.**

Output 7.32 Data Set Created by PROC MEANS with BY

AVERAGE

OBS	REGION	AVGSALES
1	EAST	22936.6
2	NORTH	26825.1
3	SOUTH	29115.0
4	WEST	27965.5

The SAS System does not support using a BY statement with more than one SET statement in the same DATA step. However, you can combine the regional averages with each observation in MATCH3 by merging the two data sets:

```
DATA NEW;
   MERGE MATCH3 AVERAGE;
   BY REGION;
```

See **Output 7.33**.

Output 7.33 Combining One Observation with All Observations in a BY Group

NEW WITH MERGING

OBS	REGION	SALESREP	JANSALES	FEBSALES	MARSALES	AVGSALES
1	EAST	GREEN	.	.	24960	22936.6
2	EAST	ISSAC	.	21991	27997	22936.6
3	EAST	STAFER	9664	.	.	22936.6
4	EAST	STRIDE	27253	29100	24700	22936.6
5	EAST	YOUNG	22969	20866	19866	22936.6
6	NORTH	FARLOW	32719	29219	21279	26825.1
7	NORTH	RICHARDS	.	11927	17927	26825.1
8	NORTH	SMITH, T	38712	31909	30909	26825.1
9	SOUTH	MOORE	25718	23312	28372	29115.0
10	SOUTH	ROCKFELT	38737	34212	29272	29115.0
11	SOUTH	STELAM	27926	25371	.	29115.0
12	WEST	CURCI	21531	18432	18432	27965.5
13	WEST	GRECO	18523	21497	20497	27965.5
14	WEST	RYAN	32915	33041	29047	27965.5
15	WEST	TOMAS	42109	38300	30300	27965.5
16	WEST	VETTER	38928	.	.	27965.5

More Uses of SET Statements

One-to-one matching Suppose that you want to read an observation from one data set and then an observation from another data set and combine them into a new observation; then read another one from the first data set, another from the second, and combine them into a second observation, and so on. New observations are formed by combining the observations in the two data sets. The statements

```
DATA NEW;
   SET A;
   SET B;
```

read an observation from A, followed by an observation from B, then output the combination to data set NEW. It is important to note that:

- SAS stops building the data set NEW when **either** A or B runs out of observations. Thus, the DATA step above differs from the step

  ```
  DATA NEW;
     MERGE A B;
  ```

  only because MERGE continues to match observations one-by-one from A and B even if one of the data sets contains no more observations. In that case, variables from the empty data set have missing values in the new combined observation.
- Variables with the same name in the two data sets have the values from B (the last data set read in the step) in the new observation.
- When variables with the same name in the two data sets have different attributes (length, label, format), the attributes of the variable in the new data set depend on the order in which the data sets are mentioned in the step. The rule is that the **length** of the variable in the new data set is the length of the variable in the first data set mentioned that contains the variable. The label and format are taken from the first data set that was

created with an explicit FORMAT or LABEL statement specification for the variable.

Reading from the same data set more than once You can read from the same data set more than once in a DATA step by giving the name of the data set in more than one SET statement. Each SET statement treats the data set independently from other SET statements and reads records sequentially from the last time **the same** SET statement was executed. If you want to **continue** reading sequentially from the same data set, you must execute the same SET statement.

For example, suppose you have a SAS data set named A. The two DATA steps below read data set A in different ways:

```
DATA B;
   SET A;
   IF condition THEN SET A;
```

To visualize how this step works, imagine that each SET statement in a DATA step makes available an entire data set from which to read. Each data set has its own pointer. In this case, having two SET statements that reference the same data set A is to SAS the same as having two data sets A: each SET statement is treated independently. The first statement causes SAS to read sequentially from the "first" data set A; when the conditional SET is executed, SAS reads from the "second" data set A.

The DATA step below differs from the previous example:

```
DATA B;
   LINK IT;
   IF condition THEN LINK IT;
   RETURN;
IT: SET A;
   RETURN;
```

Here, a LINK statement causes SAS to jump to the same SET statement so that SAS reads sequentially from data set A each time the statement labeled IT is executed.

An example of using the same data set more than once with the MERGE statement is shown in the section **Merging a Data Set with Itself**.

Direct access of SAS data sets You can access SAS data sets directly rather than sequentially by using the POINT= and NOBS= options in the SET statement. For example, suppose that you want to select every fifth sales representative from data set MASTER, described above. If you are uncertain of the total number in data set MASTER, you can use these statements:

```
DATA SUBSET;
   DO NUMBER=1 TO TOTAL BY 5;
      SET MASTER POINT=NUMBER NOBS=TOTAL;
      OUTPUT;
      END;
   STOP;
```

The value SAS assigns the variable given by the NOBS= option is the total number of observations in MASTER. In this case, the SET statement reads the observation from MASTER corresponding to the value of the POINT= variable. Compare the resulting data set SUBSET with the original data set MASTER to verify that it chose observations 1, 6, 11, and so on from MASTER. See **Output 7.34**.

Output 7.34 Observations Read by Direct Access

SUBSET

OBS	REGION	SALESREP	NAME	BEGDATE	SALES
1	EAST	GREEN	GREEN, ROBERT C.	02JAN84	551165
2	NORTH	FARLOW	FARLOW, DOUGLAS C.	08JUL85	447859
3	SOUTH	ROCKFELT	ROCKFELT, STEPHEN L.	05APR85	160566
4	WEST	RYAN	RYAN, HENRY H.	25OCT84	816871

Multiple SET statements: direct access Suppose that each of the JAN, FEB, and MAR data sets described earlier contains a variable that identifies the observation number of the sales representative in the MASTER data set. The JAN data set is shown in **Output 7.35**. The value of ID is the number of the corresponding observation in MASTER.

Output 7.35 Data Set with ID Variable

DATA SET JAN WITH ID VARIABLE

OBS	REGION	SALESREP	SALESAMT	ID
1	EAST	STAFER	9664	3
2	EAST	YOUNG	22969	5
3	EAST	STRIDE	27253	4
4	WEST	VETTER	38928	18
5	WEST	CURCI	21531	13
6	WEST	GRECO	18523	14
7	WEST	RYAN	32915	16
8	WEST	TOMAS	42109	17
9	SOUTH	ROCKFELT	38737	11
10	SOUTH	MOORE	25718	10
11	SOUTH	STELAM	27926	12
12	NORTH	FARLOW	32719	6
13	NORTH	SMITH, T	38712	9

The MASTER data set is shown in **Output 7.36**.

Output 7.36 Master Data Set Showing ID Variable and Date of Beginning Work

MASTER DATA SET WITH ID VARIABLE

OBS	REGION	SALESREP	NAME	BEGDATE	SALES	ID
1	EAST	GREEN	GREEN, ROBERT C.	02JAN84	551165	1
2	EAST	ISSAC	ISSAC, HOWARD T.	16JAN84	281910	2
3	EAST	STAFER	STAFER, MICHAEL C.	30JAN84	15458	3
4	EAST	STRIDE	STRIDE, JOHN M.	05APR84	498108	4
5	EAST	YOUNG	YOUNG, MARTIN V.	19APR84	354962	5
6	NORTH	FARLOW	FARLOW, DOUGLAS C.	08JUL85	447859	6
7	NORTH	RICHARDS	RICHARDS, BRENDA K.	22JUL85	169780	7
8	NORTH	SMITH, M	SMITH, MARY A.	11OCT85	508701	8
9	NORTH	SMITH, T	SMITH, THOMAS M.	25OCT85	748036	9
10	SOUTH	MOORE	MOORE, PATRICIA P.	30JAN85	480404	10
11	SOUTH	ROCKFELT	ROCKFELT, STEPHEN L.	05APR85	160566	11
12	SOUTH	STELAM	STELAM, MICHAEL O.	19APR85	644936	12
13	WEST	CURCI	CURCI, CYNTHIA L.	08JUL84	869282	13
14	WEST	GRECO	GRECO, ANTHONY T.	22JUL84	122341	14
15	WEST	HARRELL	HARRELL, DAVID R.	11OCT84	297837	15
16	WEST	RYAN	RYAN, HENRY H.	25OCT84	816871	16
17	WEST	TOMAS	TOMAS, JAMES M.	02JAN85	162072	17
18	WEST	VETTER	VETTER, WILLIAM A.	16JAN85	958498	18

You want to determine which employee in the JAN data set began work before January, 1985. You can read in an observation from JAN, access that observation directly from the MASTER data set using the variable ID, then check the value of BEGDATE. Consider these statements:

```
DATA CHECK;
   SET JAN;
   SET MASTER POINT=ID;
   IF BEGDATE<='01JAN85'D;
   KEEP REGION NAME;
```

The result is shown in **Output 7.37**.

Output 7.37 Observations Read by Direct Access: Two SET Statements

CHECK

OBS	REGION	NAME
1	EAST	STAFER, MICHAEL C.
2	EAST	YOUNG, MARTIN V.
3	EAST	STRIDE, JOHN M.
4	WEST	CURCI, CYNTHIA L.
5	WEST	GRECO, ANTHONY T.
6	WEST	RYAN, HENRY H.

When SAS reads from data set JAN, the value of ID in the record is the value used by the POINT= option to retrieve the correct observation from MASTER. If BEGDATE has a SAS date value corresponding to a date before January 1, 1985, the observation is selected.

Note that this step does not need a STOP statement since the DATA step stops when the JAN data set runs out of observations.

Merging a Data Set with Itself

The last example in this section merges one SAS data set with itself to create a network of connecting routes. For example, data set TRAVEL contains direct routes from several North Carolina cities to others, showing the direction, highway, and number of miles. See **Output 7.38**.

Output 7.38 Input Data Set: Merging a Data Set with Itself

TRAVEL DATA SET

OBS	FROM	TO	DRECTN	MILES	HIWAY
1	RALEIGH	WAKEFIELD	NE	18	64
2	WAKEFIELD	WILSON	SE	24	264
3	RALEIGH	CLAYTON	SE	13	70
4	CLAYTON	WILSON	NE	28	42
5	CHAPEL HILL	PITTSBORO	S	16	501
6	PITTSBORO	ASHEBORO	W	38	64
7	GREENSBORO	SANFORD	SE	34	87
8	RALEIGH	PITTSBORO	W	32	64
9	PITTSBORO	SILER CITY	W	16	64
10	SILER CITY	GREENSBORO	NW	35	421

(continued on next page)

(continued from previous page)

11	PITTSBORO	BURLINGTON	NW	28	87
12	RALEIGH	DURHAM	NW	23	70
13	DURHAM	GREENSBORO	W	54	85
14	RALEIGH	LOWES GROVE	NW	18	40
15	LOWES GROVE	CHAPEL HILL	W	11	54
16	CHAPEL HILL	BURLINGTON	NW	22	54

You want to create a new data set containing routes to additional cities, including intermediate cities. You can merge data set TRAVEL with itself by matching TO with FROM. Create differently sorted versions of TRAVEL with the SORT procedure:

```
PROC SORT DATA=TRAVEL OUT=START;
   BY TO;
PROC SORT DATA=TRAVEL OUT=FINISH;
   BY FROM;
DATA CONNECTN;
   MERGE START(IN=INSTART RENAME=(TO=VIA))
      FINISH(IN=INFINISH RENAME=(FROM=VIA DRECTN=DRECTN2
      HIWAY=HIWAY2 MILES=MILES2));
   BY VIA;
   IF INSTART AND INFINISH;
   TOTAL=MILES+MILES2;
```

Data set CONNECTN is shown in **Output 7.39**.

Output 7.39 Result of Merging Data Set with Itself

CONNECTN (ONE MERGE)

OBS	FROM	VIA	DRECTN	MILES	HIWAY	TO	DRECTN2	MILES2	HIWAY2	TOTAL
1	LOWES GROVE	CHAPEL HILL	W	11	54	PITTSBORO	S	16	501	27
2	LOWES GROVE	CHAPEL HILL	W	11	54	BURLINGTON	NW	22	54	33
3	RALEIGH	CLAYTON	SE	13	70	WILSON	NE	28	42	41
4	RALEIGH	DURHAM	NW	23	70	GREENSBORO	W	54	85	77
5	SILER CITY	GREENSBORO	NW	35	421	SANFORD	SE	34	87	69
6	DURHAM	GREENSBORO	W	54	85	SANFORD	SE	34	87	88
7	RALEIGH	LOWES GROVE	NW	18	40	CHAPEL HILL	W	11	54	29
8	CHAPEL HILL	PITTSBORO	S	16	501	ASHEBORO	W	38	64	54
9	RALEIGH	PITTSBORO	W	32	64	SILER CITY	W	16	64	48
10	RALEIGH	PITTSBORO	W	32	64	BURLINGTON	NW	28	87	60
11	PITTSBORO	SILER CITY	W	16	64	GREENSBORO	NW	35	421	51
12	RALEIGH	WAKEFIELD	NE	18	64	WILSON	SE	24	264	42

The variables TO in data set START and FROM in data set FINISH are renamed to VIA and used as the matching variable for the merge. Other variables with duplicate names are also renamed in data set FINISH so that the information in those variables is not lost in the new data set. TOTAL contains the total mileage between the starting point and the final destination.

You can repeat this process over and over. To get routes between cities with two intermediate cities, merge CONNECTN with FINISH:

```
PROC SORT DATA=CONNECTN;
   BY TO;
DATA CONNECT2;
   MERGE CONNECTN(IN=INCONN RENAME=(TO=VIA2))
```

```
      FINISH(IN=INFINISH RENAME=(FROM=VIA2 DRECTN=DRECTN3
      HIWAY=HIWAY3 MILES=MILES3));
    BY VIA2;
    IF INCONN AND INFINISH;
    TOTAL=MILES+MILES2+MILES3;
```

The result is shown in **Output 7.40**.

Output 7.40 Merging Data Set with Itself: Second Merge

CONNECT2 (TWO MERGES)

OBS	FROM	VIA	DRECTN	MILES	HIWAY	VIA2	DRECTN2	MILES2	HIWAY2	TOTAL	TO	DRECTN3	MILES3	HIWAY3
1	RALEIGH	LOWES GROVE	NW	18	40	CHAPEL HILL	W	11	54	45	PITTSBORO	S	16	501
2	RALEIGH	LOWES GROVE	NW	18	40	CHAPEL HILL	W	11	54	51	BURLINGTON	NW	22	54
3	RALEIGH	DURHAM	NW	23	70	GREENSBORO	W	54	85	111	SANFORD	SE	34	87
4	PITTSBORO	SILER CITY	W	16	64	GREENSBORO	NW	35	421	85	SANFORD	SE	34	87
5	LOWES GROVE	CHAPEL HILL	W	11	54	PITTSBORO	S	16	501	65	ASHEBORO	W	38	64
6	LOWES GROVE	CHAPEL HILL	W	11	54	PITTSBORO	S	16	501	43	SILER CITY	W	16	64
7	LOWES GROVE	CHAPEL HILL	W	11	54	PITTSBORO	S	16	501	55	BURLINGTON	NW	28	87
8	RALEIGH	PITTSBORO	W	32	64	SILER CITY	W	16	64	83	GREENSBORO	NW	35	421

Data set CONNECT2 shows that you can get from Raleigh to Pittsboro via Lowes Grove and Chapel Hill, a total distance of 45 miles.

REPORT WRITING

You can write reports that meet your specifications by using FILE and PUT statements in a SAS DATA step. If the data to appear in the report are stored in one or more SAS data sets, you use a SET, MERGE, or UPDATE statement to read the data. Or the data may be stored in an external (non-SAS) file, in which case you need an INPUT statement and an INFILE or CARDS statement in the report-writing DATA step. In either case, a FILE statement defines the output file where the report is to be written; PUT statements describe the lines in the report.

Since you are using a DATA step to do the report-writing, you have access to all DATA step features. For example,

- Options to check for end-of-file can be used to handle final processing for the report.
- Special variables set up when a BY statement is used let you check for control breaks in the data.
- Statements can be executed conditionally when a new page begins.

Suppose you want to print an annual sales report. The data you need are in SAS data set SALES. See **Output 7.41**.

Output 7.41 Input Data Set for Report

SALES

OBS	REGION	MANAGER	SALESREP	SALESAMT
1	EAST	STAFER	64	9664
2	EAST	YOUNG	89	22969
3	EAST	STRIDE	90	27253
4	EAST	ISSAC	41	21991
5	EAST	GREEN	29	23804
6	EAST	MOTTERS	82	20243
7	EAST	KRAUSE	71	18209
8	EAST	POST	55	20660
9	EAST	POWELL	10	6753
10	WEST	LARSON	57	21547
11	WEST	ARNOLD	56	35687
12	WEST	MANSTART	21	15034
13	WEST	VETTER	95	38928
14	WEST	CURCI	62	21531
15	WEST	GRECO	31	18523
16	WEST	RYAN	61	32915
17	WEST	TOMAS	84	42109
18	SOUTH	MORING	18	10343
19	SOUTH	WOOD	13	9817
20	SOUTH	GREENE	76	52516
21	SOUTH	STAYDEN	43	20354
22	SOUTH	TAYLOR	43	28722
23	SOUTH	ROCKFELT	81	38737
24	SOUTH	MOORE	49	25718
25	SOUTH	STELAM	58	27926
26	NORTH	FARLOW	49	32719
27	NORTH	SMITH, T	52	38712
28	NORTH	RICHARDS	10	11927
29	NORTH	SMITH, M	37	34709
30	NORTH	MURPHY	17	17266
31	NORTH	BARREY	62	42166

The report is to list each sales manager (variable MANAGER), the number of representatives in his district (SALESREP), and the total sales made by that district in the past year (SALESAMT). The report is to be printed in order of sales region (REGION), with subtotals for sales force and total sales for each region. Overall totals are to be printed at the end of the report. The report in its finished form is shown in **Output 7.42**.

Output 7.42 Finished Report

ANY COMPANY
REGIONAL SALES TOTALS
1985

REGION	SALES MANAGER	TOTAL SALES FORCE	TOTAL GOODS SOLD
EAST	STAFER	64	9,664
	YOUNG	89	22,969
	STRIDE	90	27,253
	ISSAC	41	21,991
	GREEN	29	23,804
	MOTTERS	82	20,243
	KRAUSE	71	18,209
	POST	55	20,660
	POWELL	10	6,753
TOTAL		531	$171,546

(continued on next page)

(continued from previous page)

NORTH	FARLOW	49	32,719
	SMITH, T	52	38,712
	RICHARDS	10	11,927
	SMITH, M	37	34,709
	MURPHY	17	17,266
	BARREY	62	42,166
	TOTAL	227	$177,499
SOUTH	MORING	18	10,343
	WOOD	13	9,817
	GREENE	76	52,516
	STAYDEN	43	20,354
	TAYLOR	43	28,722
	ROCKFELT	81	38,737
	MOORE	49	25,718
	STELAM	58	27,926
	TOTAL	381	$214,133
WEST	LARSON	57	21,547
	ARNOLD	56	35,687
	MANSTART	21	15,034
	VETTER	95	38,928
	CURCI	62	21,531
	GRECO	31	18,523
	RYAN	61	32,915
	TOMAS	84	42,109
	TOTAL	467	$226,274
	ALL REGIONS	1606	$789,452

Printing individual lines One way to begin writing the SAS program for completing the report is to consider what should be printed for each record read by the DATA step and then to handle exceptions. Observations in data set SALES correspond to district managers, and you want to print a line of information about a manager each time the DATA step is executed. The statements below print the individual manager lines:

```
DATA _NULL_;
   SET SALES;
   FILE PRINT;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
```

The DATA statement uses the special data set name _NULL_ since you do not need to create a new data set. (See the **DATA Statement** in "Statements Used in the DATA Step" for more information about _NULL_.)

The SET statement reads observations from the SAS data set SALES.

The FILE statement uses the special fileref PRINT in which lines printed by the corresponding PUT statements are printed in the procedure output file.

Each time the PUT statement is executed, SAS writes the values of MANAGER, SALESREP, and SALESAMT from the current observation. The PUT statement gives the positions and formats the values should have on the output lines.

Note: outlining the report on a coding form is a good way to determine each variable's position on the report. Do this before you begin writing the SAS program.

Printing titles on the report To execute a group of statements only when SAS begins printing a new page, use the HEADER= option of the FILE statement to give the label of the group of statements. To cancel the default SAS title or any other titles defined by earlier TITLE statements, use the NOTITLES option of the

FILE statement. The boldface additions to the program below add titles and column headings to the listing of individual lines:

```
DATA _NULL_;
   SET SALES;
   FILE PRINT HEADER=H NOTITLES;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   RETURN;
H: PUT / @19 'ANY COMPANY'
      / @15 'REGIONAL SALES TOTALS'
      / @23 '1985'
      //@34 'TOTAL' @46 'TOTAL'
      / @19 'SALES' @34 'SALES' @46 'GOODS'
      / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
      / @4 '------' @18 '-------' @34 '-----' @46 '-----'/;
   RETURN;
```

The RETURN statement before the labeled PUT statement prevents the statements following the label H from being printed for every observation. The RETURN statement after the labeled PUT statement completes the HEADER= statements; in this case, the group contains only the PUT and RETURN statements.

In this example you can code the series of hyphens as character literals or you can create character variables whose values are a series of hyphens. Then use the variables in the PUT statement.[2] (This method is useful if the literal required is long.) This DATA step then becomes

```
DATA _NULL_;
   SET SALES;
   R5=REPEAT('-',4);
   R6=REPEAT('-',5);
   R7=REPEAT('-',6);
   FILE PRINT HEADER=H NOTITLES;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   RETURN;
H: PUT / @19 'ANY COMPANY'
      / @15 'REGIONAL SALES TOTALS'
      / @23 '1985'
      //@34 'TOTAL' @46 'TOTAL'
      / @19 'SALES' @34 'SALES' @46 'GOODS'
      / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
      / @4 R6 @18 R7 @34 R5 @46 R5 /;
   RETURN;
```

The REPEAT function is described in "SAS Functions."

Accumulating totals As each observation is processed, you want to accumulate totals for regional sales force and regional total sales (TOTREP and TOTSALES) and for overall totals (ALLREP and ALLSALES). The sum statements needed to accumulate these totals appear in bold below:

```
DATA _NULL_;
   SET SALES;
   FILE PRINT HEADER=H NOTITLES;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   TOTSALES+SALESAMT;
   TOTREP+SALESREP;
```

```
   ALLSALES+SALESAMT;
   ALLREP+SALESREP;
   RETURN;
H: PUT / @19 'ANY COMPANY'
   / @15 'REGIONAL SALES TOTALS'
   / @23 '1985'
   //@34 'TOTAL' @46 'TOTAL'
   / @19 'SALES' @34 'SALES' @46 'GOODS'
   / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
   / @4 '------' @18 '-------' @34 '-----' @46 '-----'/;
   RETURN;
```

Beginning a new BY group Since the report shows managers by region, the input data should be sorted by region. A BY statement then takes advantage of the special FIRST. and LAST. byvariables, which let you check for changes in region.

When an observation is the first in the data set to have a given value of REGION, you want to reset the variables containing regional subtotals to zero before printing the observation. In addition, on the line with the first observation in the region you want to print the name of the region. The DO group shown below in bold is executed only for the first observation in a region:

```
PROC SORT DATA=SALES;
   BY REGION;
DATA _NULL_;
   SET SALES;
   BY REGION;
   FILE PRINT HEADER=H NOTITLES;
   IF FIRST.REGION THEN DO;
      TOTSALES=0;
      TOTREP=0;
      PUT @4 REGION $5. @;
      END;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   TOTSALES+SALESAMT;
   TOTREP+SALESREP;
   ALLSALES+SALESAMT;
   ALLREP+SALESREP;
   RETURN;
H: PUT / @19 'ANY COMPANY'
      / @15 'REGIONAL SALES TOTALS'
      / @23 '1985'
      //@34 'TOTAL' @46 'TOTAL'
      / @19 'SALES' @34 'SALES' @46 'GOODS'
      / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
      / @4 '------' @18 '-------' @34 '-----' @46 '-----'/;
   RETURN;
```

When the value of FIRST.REGION is 1, the current observation is the first to have the REGION value, and the statements following the DO group are executed. The PUT statement in this group of statements ends in a trailing at sign (@), so the pointer remains on the line after printing the value of region. The next PUT statement in the DATA step begins printing at that point. Normally, after a PUT statement that does not end in a trailing @ sign is executed, SAS releases the line just written on.

Ending a BY group When information for the last manager in a region has been printed, you want to print regional subtotals.

```
PROC SORT DATA=SALES;
   BY REGION;
DATA _NULL_;
   SET SALES;
   BY REGION;
   FILE PRINT HEADER=H NOTITLES;
   IF FIRST.REGION THEN DO;
      TOTSALES=0;
      TOTREP=0;
      PUT @4 REGION $5. @;
      END;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   TOTSALES+SALESAMT;
   TOTREP+SALESREP;
   ALLSALES+SALESAMT;
   ALLREP+SALESREP;
   IF LAST.REGION THEN
      PUT @33 '------' @42 '---------'
         / @10 'TOTAL' @33 TOTREP 5.  @42 TOTSALES DOLLAR9. //;
   RETURN;
H: PUT / @19 'ANY COMPANY'
      / @15 'REGIONAL SALES TOTALS'
      / @23 '1985'
      //@34 'TOTAL' @46 'TOTAL'
      / @19 'SALES' @34 'SALES' @46 'GOODS'
      / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
      / @4 '------' @18 '-------' @34 '-----' @46 '-----'/;
   RETURN;
```

The PUT statement in this new group of statements prints a line containing dashes under the TOTAL SALES FORCE and TOTAL GOODS SOLD columns and then prints the word TOTAL and the TOTREP and TOTSALES values using the DOLLAR. format.

Printing grand totals After information for all regions is printed, you want to print the total sales force and total sales values for all regions. The END= option in the SET statement lets you define a variable whose value is 1 when the current observation is the last from SALES. When this variable's value is 1, print the overall totals on the report.

```
PROC SORT DATA=SALES;
   BY REGION;
DATA _NULL_;
   SET SALES END=EOF;
   BY REGION;
   FILE PRINT HEADER=H NOTITLES;
   IF FIRST.REGION THEN DO;
      TOTSALES=0;
      TOTREP=0;
      PUT @4 REGION $5. @;
      END;
   PUT @14 MANAGER $15. @35 SALESREP 3. @45 SALESAMT COMMA6.;
   TOTSALES+SALESAMT;
   TOTREP+SALESREP;
   ALLSALES+SALESAMT;
```

```
   ALLREP+SALESREP;
   IF LAST.REGION THEN
      PUT @33 '------' @42 '---------'
         / @10 'TOTAL' @33 TOTREP 5.  @42 TOTSALES DOLLAR9. //;
   IF EOF THEN PUT / @10 'ALL REGIONS'
      @32 ALLREP 6. @41 ALLSALES DOLLAR10.;
   RETURN;
H: PUT / @19 'ANY COMPANY'
      / @15 'REGIONAL SALES TOTALS'
      / @23 '1985'
      //@34 'TOTAL' @46 'TOTAL'
      / @19 'SALES' @34 'SALES' @46 'GOODS'
      / @4 'REGION' @18 'MANAGER' @34 'FORCE' @46 'SOLD'
      / @4 '------' @18 '-------' @34 '-----' @46 '-----'/;
   RETURN;
```

Whole-page access: the N=PS option Normally, when you print a report, you have access to one line at a time. Once you execute a PUT statement that prints on that line, you cannot return to the line for later printing. This poses a problem for printing reports containing multiple columns of information, among others. The N=PAGESIZE or N=PS option of the FILE statement tells SAS to hold an entire page in the output buffer until the page is full or until you tell SAS to release it. This means that for multiple columns, you can print a column at a time; when PUT statements have printed one column, you can return to the top of the next column to continue printing.

Suppose that you want to print a listing of district managers from the data set above. The multicolumn list is to show the rank order of salesmen based on their district's total sales for the previous year.

The SAS program below first sorts the SALES data set in descending order of sales and then prints the two-column report:

```
OPTIONS LINESIZE=72 NODATE;
PROC SORT DATA=SALES;
   BY DESCENDING SALESAMT;
TITLE 'ANY COMPANY';
TITLE2 'SALES MANAGERS FOR 1985';
TITLE3 'RANKED BY TOTAL SALES';
DATA _NULL_;
   FILE PRINT N=PS HEADER=COLUMNS;
   R32=REPEAT('-',31);
   DO C=2,36;
      DO L=6 TO 25;
         SET SALES;
         PUT #L @C
            MANAGER $10. +1 '(' REGION $5. ')' +5 SALESAMT DOLLAR8.;
         END;
      END;
   PUT _PAGE_;
   RETURN;
COLUMNS: PUT / @2 'MANAGER' +4 '(REGION)' @24 '''85 SALES'
             @36 'MANAGER' +4 '(REGION)' @58 '''85 SALES'
            / @2 R32 @36 R32;
    RETURN;
```

The job above uses both TITLE statements and the HEADER= option of the FILE statement to produce table headings. When TITLE statements are used, PUT

statements that are part of a header group of statements print in the "window" remaining after all titles are printed.

The two DO loops in the job represent column and line counters. The variable C, which is the index for the outer DO loop, becomes the value of the horizontal print position where each column of information begins. L, the index for the inner loop, determines the line, or vertical print position. Printing of each column begins at line 6.

PUT _PAGE_ tells SAS to release the output to the print file specified by the FILE statement when both DO loops have been satisfied. SAS then returns to the top of the DATA step for a new execution and a new page. This process continues until all observations in SALES have been read. PUT _PAGE_ also releases the page when the DATA step is complete.

Notice in the report shown in **Output 7.43** that only part of the second column is needed to print the SALES data set in this format.

Output 7.43 Multicolumn Report with Partly Filled Column

```
                          ANY COMPANY
                    SALES MANAGERS FOR 1985
                     RANKED BY TOTAL SALES

MANAGER    (REGION)   '85 SALES   MANAGER    (REGION)   '85 SALES
---------------------------------  ---------------------------------


GREENE     (SOUTH)      $52,516   STAYDEN    (SOUTH)      $20,354
BARREY     (NORTH)      $42,166   MOTTERS    (EAST )      $20,243
TOMAS      (WEST )      $42,109   GRECO      (WEST )      $18,523
VETTER     (WEST )      $38,928   KRAUSE     (EAST )      $18,209
ROCKFELT   (SOUTH)      $38,737   MURPHY     (NORTH)      $17,266
SMITH, T   (NORTH)      $38,712   MANSTART   (WEST )      $15,034
ARNOLD     (WEST )      $35,687   RICHARDS   (NORTH)      $11,927
SMITH, M   (NORTH)      $34,709   MORING     (SOUTH)      $10,343
RYAN       (WEST )      $32,915   WOOD       (SOUTH)       $9,817
FARLOW     (NORTH)      $32,719   STAFER     (EAST )       $9,664
TAYLOR     (SOUTH)      $28,722   POWELL     (EAST )       $6,753
STELAM     (SOUTH)      $27,926
STRIDE     (EAST )      $27,253
MOORE      (SOUTH)      $25,718
GREEN      (EAST )      $23,804
YOUNG      (EAST )      $22,969
ISSAC      (EAST )      $21,991
LARSON     (WEST )      $21,547
CURCI      (WEST )      $21,531
POST       (EAST )      $20,660
```

Unequal column sizes The last example shows another multicolumn report using the SALES data set. Each column in the report corresponds to a different region. Since all regions do not have the same number of district managers, the columns contain different numbers of observations.

```
PROC SORT DATA=SALES;
   BY REGION DESCENDING SALESAMT;
OPTIONS LINESIZE=120 NODATE;
TITLE 'ANY COMPANY';
TITLE2 'SALES MANAGERS FOR 1985';
TITLE3 'RANKED BY TOTAL SALES WITHIN REGION';
DATA _NULL_;
   FILE PRINT N=PS;
   R24=REPEAT('-',23);
   DO C=2, 29, 56, 84;
      DO L=6 TO 25;
         SET SALES;
```

```
            BY REGION;
            IF FIRST.REGION THEN LINK HEADING;
            PUT #L @C MANAGER $12. +3 SALESAMT DOLLAR8.;
            IF LAST.REGION THEN GO TO NEWCOL;
            END;
NEWCOL: END;
   RETURN;
HEADING: PUT #1 / @C +8 REGION / @C R24 /
      @C +2 'MANAGER' @C +14 '''85 SALES'/;
   RETURN;
```

The report is shown in **Output 7.44**.

Output 7.44 Multicolumn Report with Unequal Column Lengths

ANY COMPANY
SALES MANAGERS FOR 1985
RANKED BY TOTAL SALES WITHIN REGION

EAST		NORTH		SOUTH		WEST	
MANAGER	'85 SALES	MANAGER	'85 SALES	MANAGER	'85 SALES	MANAGER	'85 SALES
STRIDE	$27,253	BARREY	$42,166	GREENE	$52,516	TOMAS	$42,109
GREEN	$23,804	SMITH, T	$38,712	ROCKFELT	$38,737	VETTER	$38,928
YOUNG	$22,969	SMITH, M	$34,709	TAYLOR	$28,722	ARNOLD	$35,687
ISSAC	$21,991	FARLOW	$32,719	STELAM	$27,926	RYAN	$32,915
POST	$20,660	MURPHY	$17,266	MOORE	$25,718	LARSON	$21,547
MOTTERS	$20,243	RICHARDS	$11,927	STAYDEN	$20,354	CURCI	$21,531
KRAUSE	$18,209			MORING	$10,343	GRECO	$18,523
STAFER	$9,664			WOOD	$9,817	MANSTART	$15,034
POWELL	$6,753						

This example is similar to the one above except that instead of the HEADER= option in the FILE statement, the job uses a LINK-RETURN group for column titles. It links to a labeled statement outside the DO loop each time the REGION value changes. When the last manager for a region has been printed, the program jumps to the END statement for the outer DO loop. This causes SAS to increment the column counter and begin printing a new column.

NOTES

1. **AOS/VS, PRIMOS, and VMS:** You can use multidimensional array subscripting to process these data values. This DATA step reads the data and creates SAS data set RESULTS2:

```
DATA RESULTS2;
   INFILE IN MISSOVER;
   ARRAY PRODUCT(5,5) ITEM1-ITEM25;
   INPUT ID 1-3 @;
   DO UNTIL(J=. OR J*I=25);
      INPUT J @;
      IF J¬=. THEN INPUT I PRODUCT(J,I) @;
      END;
   KEEP ID ITEM1-ITEM25;
```

The ARRAY statement defines a two-dimensional array with five rows and five columns. Variables ITEM1-ITEM5 form the first row, ITEM6-ITEM10 the second row, and so on.

The first INPUT statement in the DO UNTIL loop reads a value for J. As before, J is missing when there are no more values on the data line. Otherwise, the next INPUT statement reads a value for I and uses that along with the value for J to determine which variable in array PRODUCT is given the value.

CMS, OS, VM/PC, and VSE: You can nest implicitly subscripted arrays (also called creating an "array of arrays") to achieve the effect of multidimensional implicitly subscripted arrays. This DATA step reads the data and creates SAS data set RESULTS2:

```
DATA RESULTS2;
   INFILE IN MISSOVER;
   ARRAY ONE ITEM1-ITEM5;
   ARRAY TWO ITEM6-ITEM10;
   ARRAY THREE ITEM11-ITEM15;
   ARRAY FOUR ITEM16-ITEM20;
   ARRAY FIVE ITEM21-ITEM25;
   ARRAY PRODUCT (J) ONE TWO THREE FOUR FIVE;
   INPUT ID 1-3 @;
   DO UNTIL(J=. OR J*_I_=25);
      INPUT J @;
      IF J¬=. THEN INPUT _I_ PRODUCT @;
      END;
   KEEP ID ITEM1-ITEM25;
```

Six arrays are defined in the DATA step. The first five contain as elements the items from the questionnaire corresponding to the five products. The sixth array contains, as elements, the arrays defined above it.

The first five arrays use as their index variable the automatic variable _I_. It is possible to use the same index variable for several arrays **as long as the arrays have the same number of elements**. Note that the variables defined for each of these product arrays have names ending with a numeric suffix. This suffix has no relationship to the value of the index variable, which ranges from 1 to 5.

The first INPUT statement in the DO loop reads a value for J. As before, J is missing when there are no more values on the data line. Otherwise, J's value determines to which of the product arrays the next two values refer. The next INPUT statement reads a value for _I_, which references the item number in the appropriate product array. When the array name PRODUCT appears in the same INPUT statement, the current values of J and _I_ determine which variable is given the value.

2. **CMS, OS, VM/PC, and VSE:** You can also write the series of hyphens as *n**'-', as in 4*'-'.

THE PROC STEP

Introduction to the PROC Step

SAS® Statements Used in the PROC Step

PROC Step Applications

Chapter 8

Introduction to the PROC Step

Once you have created a SAS data set with a DATA step, you can analyze and process it with SAS procedures. SAS procedures are programs that read SAS data sets, compute statistics, print results, and create other SAS data sets. In a DATA step you can do your own programming using SAS statements to manipulate your data and to describe the SAS data set being created. In a PROC step, you call a procedure by its name—the program is already written for you.

What Is a PROC Step?

A PROC step is a group of one or more SAS statements that begins with a PROC statement.

In the simplest PROC step,

- you want to process the most recently created SAS data set.
- you want all the variables processed and computations performed on all numeric variables.
- you want the entire data set processed at once rather than in subsets.

Since the SAS System handles this situation automatically, your PROC step is only a PROC statement to name the procedure you want:

```
PROC program;
```

For other analyses, you can include options in the PROC statement, or you can add other statements and their options to the PROC step to specify the analysis you want in more detail.

For example, to process a data set other than the most recently created one, you can specify its name in the DATA= option in the PROC statement:

```
PROC program DATA=SASdataset;
```

You can also add other statements to your PROC step to specify that the data should be processed in a special way. If you add a BY statement, for example:

```
PROC program;
   BY variables;
```

the data are processed in BY groups.

PROC Step Statements

The SAS statements that can appear in a PROC step are procedure information statements and variable attribute statements. Other statements also available for use within a PROC step are described in "SAS Statements Used Anywhere."

The statements that are shared by a number of SAS procedures are introduced here; many other statements are unique to different procedures. All procedure

information statements that can be used with each procedure are also explained in detail in the individual procedure descriptions. Statements are available for all operating systems unless specifically listed. Commonly used procedure information statements are

BY specifies that the input data set is to be processed in groups defined by the BY variables.

Operating systems: All

CLASS identifies any classification variables in the analysis.

Operating systems: All

FREQ identifies a variable that represents frequency of occurrence.

Operating systems: All

ID specifies one or more variables whose values identify observations in the printed output or SAS data set created by the procedure.

Operating systems: All

MODEL specifies the variables that represent the dependent variables and independent effects for a model.

Operating systems: All

OUTPUT gives information about any output data set created by the procedure.

Operating systems: All

PARMCARDS precedes lines (parmcard lines) that the SAS System places into a temporary file before using them in the procedure.

Operating systems: CMS, OS, VM/PC, VSE

PARMCARDS4 precedes parmcard lines containing semicolons.

Operating systems: CMS, OS, VM/PC, VSE

VAR identifies the variables to be analyzed by the procedure.

Operating systems: All

WEIGHT specifies a variable whose values are relative weights for the observations.

Operating systems: All

Commonly used variable attribute statements are

ATTRIB specifies any of the attributes format, informat, label, and length for variables.

FORMAT specifies formats for printing variable values.

Operating systems: All

LABEL associates descriptive labels with variable names.

Operating systems: All

The variable attribute statements INFORMAT and LENGTH, described in "SAS Statements Used in the DATA Step," can also be used with a few SAS procedures.

You can use the DROP= and KEEP= data set options to identify variables to be excluded from or included in a data set or analysis. These options are available in the PROC step wherever data set names are used; see "SAS Files" for descriptions of these options. DROP and KEEP statements are no longer available in the PROC step.

Chapter 9

SAS® Statements Used in the PROC Step

ATTRIB Statement

Operating systems: All

The ATTRIB statement in the PROC step allows you to specify the format, informat, label, and length of one or more variables in a single statement. An attribute specified in a PROC step is associated with the variable for that PROC step and in any output data sets the procedure creates that contain the variable.

The form of the ATTRIB statement is

ATTRIB *variable* [**FORMAT=***format*][**INFORMAT=***informat*] [**LABEL=***'label'*] [**LENGTH=**[**$**]*length*] ...**;**

You can specify the following terms in the ATTRIB statement:

variable
: names the variable to receive the attributes. *Variable* can be a single variable name, a list of names, or one of the variable lists described in "Introduction to the SAS Language," except for the special lists _NUMERIC_, _CHARACTER_, and _ALL_.

FORMAT=*format*
: specifies the format to associate with the preceding variables. The format can be either a SAS format or a format you have created with the FORMAT procedure. See the FORMAT statement later in this chapter for more information on the format attribute, and see "SAS Informats and Formats" for descriptions of SAS formats.

INFORMAT=*informat*
: specifies the informat to associate with the preceding variables. See the INFORMAT statement later in this chapter for information on the informat attribute, and see "SAS Informats and Formats" for descriptions of informats.

LABEL=*'label'*
: specifies a label to be associated with the preceding variables. See the LABEL statement later in this chapter for information on variable labels.

LENGTH=[$]*length*
: specifies the length of the preceding variables. If the variables are character variables, put a dollar sign ($) in front of the length.

You can also assign variable attributes with FORMAT, INFORMAT, LABEL, and LENGTH statements. Any attribute that you have assigned with an ATTRIB statement can be changed in an individual statement and vice versa.

Here are examples of valid ATTRIB statements:

```
ATTRIB X Y Z LENGTH=$4 LABEL='SAMPLE VARIABLES';
ATTRIB HOLIDAY INFORMAT=MMDDYY. FORMAT=WORDDATE.;
ATTRIB MONTH1-MONTH12 LENGTH=3;
ATTRIB DAY1-DAY7 LABEL='DAY OF WEEK'
       WEEK1-WEEK4 LABEL='WEEK OF MONTH'
       MONTH1-MONTH12 LABEL='MONTH OF YEAR'
       SALES INFORMAT=COMMA8.2 FORMAT=DOLLAR10. LABEL='TOTAL SALES';
```

BY Statement

Operating systems: All

You can use a BY statement in a PROC step when you want the SAS System to process the data set in groups. A BY statement is always used with the SORT procedure to define the order in which the data set should be sorted. When a BY statement is used with most other procedures that analyze SAS data sets, the procedure processes each BY group separately. See the BY statement description in "Statements Used in the DATA Step" for a discussion of BY groups.

The BY statement has the form:

BY [**DESCENDING**] *variable* ... [**NOTSORTED**];

where

variable
: names the variable or variables that define the BY groups. The procedure processes the data set in the groups defined in the BY statement.

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 with the same BY values are grouped together, but the BY values are not necessarily sorted in alphabetical or numerical order. It may appear anywhere in the BY statement. The NOTSORTED option cannot be used with the SORT procedure.

If you do not specify either DESCENDING or NOTSORTED, SAS assumes that the data set is sorted in ascending order of the BY variables.

For example, the statement

```
BY DESCENDING X Y;
```

specifies that data set is sorted in descending order of the values of X and, within each X value, in ascending order of Y. The statement

```
BY DESCENDING X DESCENDING Y;
```

specifies that the data set is sorted in descending order of the values of X, and within each X value, in descending order of Y.

In the following example, data set CLASS contains observations with a variable DAY whose values are the three-letter abbreviation for the day of the week. The observations with the same DAY values are grouped together in the data set, but the DAY values are in calendar order, not alphabetical order. The following statement can be used with a PROC statement to process the data set by DAY:

```
BY DAY NOTSORTED;
```

When a BY statement is used with a procedure to process the data in BY groups, the data set being processed need not have been previously sorted by the SORT procedure. However, the data set must be in the same order as though the SORT procedure had sorted it unless NOTSORTED is specified.

For example, below is a listing of SAS data set DEGREES. The data set is sorted by STATE, CITY within STATE, and MONTH within CITY.

STATE	CITY	MONTH	DEGDAYS
NC	CHARLOTTE	1	716
NC	CHARLOTTE	2	588
NC	CHARLOTTE	3	461
NC	RALEIGH	1	760
NC	RALEIGH	2	638
NC	RALEIGH	3	502
VA	NORFOLK	1	760
VA	NORFOLK	2	661
VA	NORFOLK	3	532
VA	RICHMOND	1	853
VA	RICHMOND	2	717
VA	RICHMOND	3	569

With the data set sorted in this order, the following statements cause SAS to print two listings—a separate listing for each state:

```
PROC PRINT;
   BY STATE;
```

(The PRINT procedure has several options that allow enhanced BY-group processing when a BY statement is in effect. See the PRINT procedure description for details.)

The following statements produce descriptive statistics in four separate reports—one for each combination of STATE and CITY values:

```
PROC MEANS;
   BY STATE CITY;
   VAR DEGDAYS;
```

CLASS Statement

Operating systems: All

The CLASS (or CLASSES) statement is used by several SAS procedures to identify classification variables by which the analysis is to be performed.

The form of the CLASS statement is:

CLASS *variables*;

or

CLASSES *variables*;

where

variables specifies the classification variable(s) in the analysis. When more than one classification variable is allowed by the procedure, any form of variable list is allowed as described in "Introduction to the SAS Language."

FORMAT Statement

Operating systems: All

Introduction

You can use the FORMAT statement to associate formats with variables in PROC steps. The formats can be either SAS formats or formats you have defined with PROC FORMAT. You can give the same format to several variables or different formats to different variables with a single FORMAT statement. When the SAS System prints values of the variables, it uses the associated format to print the values.

A format specified in a PROC step is associated with the variable for the duration of that PROC step and in any output data sets the procedure creates that contain the variable.

You can associate formats with variables with the statement

FORMAT *variables* [*format*] ...**;**

These terms are included in the FORMAT statement:

variables names the variable or variables you want to associate with a format.

format gives the format you want SAS to use for writing values of the variable or variables in the previous variable list. Every format name ends with a period (for example, SEXFMT.) or has a period between the width value and number of decimal places (for example, DOLLAR8.2).

See "SAS Informats and Formats" for more information on the types of SAS formats that are available. See "The FORMAT Procedure" for information on how to define your own output formats.

If a variable appears in more than one FORMAT statement, the format given in the last FORMAT statement is used. A FORMAT statement used in a DATA step to associate one format with a variable can be overridden by a FORMAT statement in a later PROC step. The original format name remains stored with the variable in the data set. To disassociate a format from a variable, use the variable's name in a FORMAT statement with no format.

You can also assign formats with the ATTRIB statement. You can change a format assigned in a FORMAT statement with an ATTRIB statement and vice versa.

If you have used the FORMAT procedure to define your own formats, you need a FORMAT statement to associate the format with one or more variables:

```
PROC FORMAT;
   VALUE SEXFMT 1='MALE'
                2='FEMALE';
PROC PRINT DATA=ALL;
   FORMAT SEX SEXFMT.;
```

The FORMAT procedure defines the SEXFMT. format. The FORMAT statement in the PROC step associates SEXFMT. with SEX, a variable in data set ALL. When

the values of SEX are printed by the PRINT procedure, MALE and FEMALE are printed instead of the numbers 1 and 2.

To use some SAS procedures like FREQ and GLM, you may need to divide observations on continuous variables into distinct groups or categories. You can use the FORMAT procedure to define these categories or ranges of values. If you then use a FORMAT statement to associate the new format defined by PROC FORMAT with the variables, FREQ or GLM uses the formatted values to determine the category or group into which each observation falls. See "PROC Step Applications" for an example using the FORMAT statement with the FREQ procedure.

Using FORMAT for Datetime Values

When you are using SAS datetime values in procedures, you must assign them a corresponding date, time, or datetime format in order for the values to be printed in an intelligible form. If you want to associate the format with the variable only for the duration of the procedure, use a FORMAT statement in the PROC step. For example,

```
PROC PRINT;
   VAR DATE FARENHT CELSIUS;
   FORMAT DATE DATE7.;
   TITLE 'HIGH TEMPERATURE FOR DAY';
```

The result of the PRINT procedure is a listing of each date in the data set in the form *ddMMMyy* (DATE7.) with the Fahrenheit and Celsius temperature values. Without the FORMAT statement, values of DATE appear as the number of days between January 1, 1960, and the value of the variable DATE, a large number that is difficult to interpret. See "SAS Informats and Formats" for more information on SAS date, time, and datetime values.

FREQ Statement

Operating systems: All

The FREQ statement is used with several procedures to identify a variable which represents the frequency of occurrence for the other values in the observation.

The form of the FREQ statement is:

FREQ *variable*;

where

variable represents the frequency of occurrence for the observation.

When a FREQ statement appears, the procedure treats the data set as though each observation appears *n* times, where *n* is the value of the FREQ variable for that observation.

If the value of the FREQ variable in an observation is less than one, the observation is not used in the analysis. If the value is not an integer, only the integer portion is used.

ID Statement

Operating systems: All

The ID statement is used by several SAS procedures to specify one or more variables whose values identify observations in the printed output or SAS data set created by the procedure.

The form of the ID statement is:

ID *variables*;

where

variables specifies the identifying variables.

For example, when an ID statement is used with the PRINT procedure, the observations are identified by the value of the ID variable; the observation number is not printed. See the census data example in "PROC Step Applications" for output showing the effect of an ID variable.

LABEL Statement

Operating systems: All

You can use a LABEL statement in a PROC step to give labels to variables. Most SAS procedures use these variable labels in writing the results of analyses. The LABEL statement is an attribute statement that can occur anywhere among the statements in a PROC step. Although the statement is available to all procedures that read SAS data sets, it is not described in any procedure description. When a LABEL statement is used in a PROC step, the labels are associated with the variables for the duration of the procedure step.

The form of the LABEL statement is

LABEL *variable='label ...';*

where

variable names the variable to be labeled.

label specifies a label of up to forty characters, including blanks, for the variable. The label must be enclosed in either single or double quotes. (If double quotes are used, the SAS system option DQUOTE must be in effect.) When single quotes are part of the label, they must be written as two single quotes. When two single quotes are used within a label to represent one single quote, they are counted as one character.

Any number of variable names and labels can appear. Here is an example:

```
PROC PLOT;
   PLOT X*Y;
   LABEL X='RESPONSE TIME' Y='HOUR OF DAY';
```

In this example, labels rather than variable names are used to label the vertical and horizontal axes of the plot.

MODEL Statement

Operating systems: All

The MODEL statement is used by several SAS statistical procedures described in the *SAS User's Guide: Statistics* to identify the model to be analyzed.

Although the form of the MODEL statement depends on the procedure used, the general form of the MODEL statement is:

MODEL *dependents=independenteffects /[options]*;

where

dependents
specifies the dependent variable or variables in the analysis.

independenteffects
specifies the independent variables or regressors in the analysis.

options
specifies one or more MODEL statement options. See the individual procedure descriptions for a list of options available with that procedure.

OUTPUT Statement

Operating systems: All

The OUTPUT statement is used by several SAS procedures to tell the procedure to create a SAS data set as output.

Although the form of the OUTPUT statement may vary with each procedure, the general form is:

OUTPUT [OUT=*SASdataset*] [*keyword*=*names*]...**;**

where

SASdataset
: specifies the name of the new SAS data set to be created by the procedure.

keyword=*names*
: names the output variables on the data set associated with keywords. Keywords vary with each procedure but are usually descriptive of a statistic or other value being output to the new data set.

For example, in the MEANS procedure step below:

```
PROC MEANS;
   VAR X;
   OUTPUT OUT=OUTMEAN MEAN=MEANX;
```

the MEAN= keyword specifies that the mean of X, calculated by the MEANS procedure, should be given the name MEANX in the new data set OUTMEAN.

PARMCARDS and PARMCARDS4 Statements

Operating systems: CMS, OS, and VM/PC

The PARMCARDS or PARMCARDS4 statement is used to supply information to a PROC step. The statement causes the SAS System to write the lines following the statement to a temporary file (the parmcards file) and read the lines into the current PROC step from there. The PARMCARDS statement is used in the BMDP procedure, in some procedures in the SUGI Supplemental Library such as the EXPLODE procedure, and in some user-written SAS procedures. The type of information supplied in parmcard lines is described with each procedure that uses the PARMCARDS statement.

The form of the PARMCARDS statement is

PARMCARDS;
parmcard lines
;

SAS recognizes the end of the parmcard lines when it sees a semicolon. The first line after the last parmcard line should be either a null statement (a line containing a single semicolon) or another SAS statement ending with a semicolon on the same line.

If the lines following the PARMCARDS statement happen to contain semicolons, you must substitute the PARMCARDS4 statement for the PARMCARDS statement. The form of the PARMCARDS4 statement is

PARMCARDS4;
data lines
;;;;

The first line after the last parmcard line should consist of four semicolons in columns 1-4 to signal the end of the parmcard lines.

PROC Statement

Operating systems: All

The PROC or PROCEDURE statement is used to begin a PROC step and to identify the procedure you want to use.

The form of the PROC statement is

PROC *program* [*options*];

where

program names the program you want to use. Normally, the name you specify is the name of a SAS procedure. You can, however, call your own program from the SAS System in a PROC statement (see below for instructions).

options specifies one or more options for the procedure. See the individual procedure descriptions for options that can be specified in the PROC statement.

Three kinds of options are commonly used:

keyword
: single keyword to request a feature of the procedure.

keyword=value
: keyword and value, where value is a number or character string.

keyword=SASdataset
: specifies an input or output SAS data set.

The options that can be specified with each procedure are described in detail in the individual procedure description.

Using other programs with SAS software Unless disallowed by your installation, you can call any program from SAS by putting its name in a PROC statement. For example, if you want to call a program named RECORDS in a SAS job, you can use the SAS statement

```
PROC RECORDS;
```

Your program will then be executed. When you are using this feature, keep in mind these facts:

- The library containing your program must be available to the system or defined in the control language for your job. See the "Operating System Notes" appendix for a discussion of this topic for your environment.
- You can check the return code issued by your program by adding a CC= option with the maximum acceptable return code to your PROC statement:

  ```
  PROC RECORDS CC=4;
  ```

  If the return code issued by your program is greater than the CC value and you are not executing SAS interactively, SAS sets OBS=0 and enters syntax check mode. (See "SAS Log and Procedure Output" for more about syntax check mode.)

VAR Statement

Operating systems: All

The VAR (or VARIABLES) statement is used by several SAS procedures to identify the variables to be analyzed.
The form of the VARIABLES statement is:

VAR *variables*;

or

VARIABLES *variables*;

where

variables identifies the variables in the data set you want analyzed by the procedure. Any valid form of variable list may be used. (See "Introduction to the SAS Language" for details.)

WEIGHT Statement

Operating systems: All

The WEIGHT statement is used by several SAS procedures to specify a variable whose values are relative weights for the observations.

The form of the WEIGHT statement is:

WEIGHT *variable*;

where

variable names the variable whose values contain the relative weights.

The WEIGHT statement is often used in analyses where the variance associated with each observation is different and the values of the weight variable are proportional to the reciprocals of the variances.

The WEIGHT statement should not be confused with the FREQ statement, which identifies a variable that represents frequency of occurrence for the observation. See the FREQ statement description for more information.

Chapter 10

PROC Step Applications

Operating systems: All

Census Data: Example 1

This example applies several SAS procedures for descriptive statistics to U.S. census data. The data are selected characteristics of U.S. cities with populations over 50,000 according to the *County and City Data Book, 1983*. The raw data are read into a SAS data set and then prepared for analysis with a variety of DATA step techniques.

Data set CCDB83 The statements in the first DATA step below read in data from an external file labeled CENSUS into a SAS data set called CCDB83. The output data set contains 418 cities as observations with data on a number of variables related to choice of retirement community. See the appendix "Operating System Notes" for examples of control language to use when reading data from an external file.

The PUT _INFILE_ statement prints the raw data for the first five observations. The file contains city records for all U.S. cities and some state and national summary records. An IF-THEN statement is used with a DELETE statement to delete the summary records which have values of 0000 for the PLACE variable. Values of zero for ELECTRIC, CITYGOVT, and CRIME are set to missing (.) using IF-THEN statements. A subsetting IF statement is used to select only cities with populations greater than or equal to 50,000. The LIST statement lists only one of the first five observations, (BIRMINGHAM), that is included in the data set being built.

Using an array named CHECK, SAS checks the variables to be analyzed for missing values (.) and values greater than 100,000,000 (codes that should not be included in the analysis).

A combination of IF-THEN/ELSE statements divides the cities into three groups based on the size of the population. A LABEL statement provides variable labels.

```
DATA CCDB83;
   INFILE CCDB83;
   ARRAY CHECK(10) POP1980 OVER65 INCOME HOUSING
      ELECTRIC CITYGOVT CRIME JANTEMP JULYTEMP RAINFALL;
   DROP I;
   INPUT
        STATE          1-2
        PLACE          3-6
        CITY   $      17-46
        POP1980       71-79
        OVER65       202-211
        INCOME      1225-1234
        HOUSING      488-497
        ELECTRIC    1929-1938 2
        CITYGOVT    1500-1509
        CRIME        422-431
        JANTEMP     1951-1960 1
        JULYTEMP    1962-1971 1
        RAINFALL    1940-1949 2 ;

   IF _N_<=5 THEN PUT _INFILE_;
   IF PLACE=0000 THEN DELETE;
   IF POP1980>=50000;
   IF _N_<=5 THEN LIST;
   IF CRIME=0 THEN CRIME=.;
   IF ELECTRIC=0 THEN ELECTRIC=.;

   DO I=1 TO 10;
      IF CHECK(I)¬=. AND CHECK(I)<100000000;
      END;

   IF 50000<POP1980<=75000 THEN POPGRP=1;
   ELSE IF 75000<POP1980<=100000 THEN POPGRP=2;
   ELSE IF POP1980>100000 THEN POPGRP=3;

   LABEL POP1980='1980 POPULATION ESTIMATE'
         OVER65='PERSONS 65 & OVER (1980)'
         INCOME='PER CAPITA MONEY INCOME (1979)'
         HOUSING='HOUSING UNITS (1980)'
         ELECTRIC='TYPICAL RESID. ELECTRIC BILL/MO. (1982)'
         CITYGOVT='FORM OF CITY GOVERNMENT (1982)'
         CRIME='CRIME RATE/100,000 POP (1981)'
         JANTEMP='MEAN JANUARY TEMPERATURE (DEGREES F)'
         JULYTEMP='MEAN JULY TEMPERATURE (DEGREES F)'
         RAINFALL='ANNUAL PRECIPITATION';
```

Output 10.1 Printout of First Five Data Observations

```
NOTE: INFILE CCDB83 IS:

      DSNAME=CCDB83.CITY,
      UNIT=TAPE,VOL=SER=UU9402,LABEL=2,DISP=OLD,
      DCB=(BLKSIZE=20150,LRECL=2015,RECFM=FB)
010000          ALABAMA                    0000005076700003444354000038938885000000000000000000077000020223540000 2873
289000009962830000009304000001066000000339230000389388800000296412000008658360000044001500002731640000000002930000007
799100001341856000000002840000103888100000845944000001609330000027363200003076255000000635030000001309600000000163000000
344570000000008800000000147000000253660000000065100000186448000000048010000016857700000009868000000504600000003991000000
281975000014673740000112023900001450011000001081550000145075500001108587000001323450000102533500000056463000001679470000
022447200000331643000002371170000134185600000941219000000002950000003390000000400637000000001880000005629400000072668000
013418560000087148200000346994000000070180000000098400001506232000010505390000035193200000020519000008067250000030763300
002217315000012524750000027006300001078971000005982770000026135700000069291000000198150000018714000001665352000016347430
000068906900000122815000002909970000009638500001511928000003949130000029034600000286707000000000300000000071000017 13000
000002470000000014547600000539977000010425710000134237100000402900000025363800000106640000000376130000003594900000013 66
950000000000000000163470000000589400001042571000001573550000015427200000061204000007199050000381301400000279074000011478
755000000000050000000005000000000050000000005000000000050000000005000000000050000000005000000000050000000005000000000050000000
000500000000050000000005000000000050000000005000000000050000000005000000000050000000005000000000050000000005000000000061299000000
586300000001855000000840590000000344000000037732000000027300000000525200000026976000002101010000000571300014185813000081
021050000006354800000705379000000315820000002158300010164414000001811560000114666000010704902000036908270000243998900001
307723000006914250000081044300000021878000016957900000015519400000277470000001062110000154810600000009950000000681010000
051065400000004122500000000005000000000050000000005000000000050000000005000000000050000000000
0100600450      ANNISTON, AL               0000000002100000031533000000295230000000081300000001426000000159540000001 7
361000000118990000000003200000000112000000003440000002952300000002323000000062230000000419400000020977000000003190000000
036800000010953000000002660000000787100000005833000000017850000000284100000023000000000007570000000019300000000256000000
004520000000015300000000002000000004220000000142900000004195000000140060000000359000000000042000000000420000000000000000
002263000000011826000000011089000000118220000000869000000118260000009215000000013880000000682800000000289000000006020000
000093400000001773000000030930000001095300000006577000000002730000002950000000004376000000001630000000021500000000617000
000109530000000963600000001275000000000000000000211800000010453000000075060000000222800000000113000000098540000000324500
000017798000000099770000002697000000746200000004378000000017880000000590000000017300000001002000000011895000000114690
000000535300000001207000000028330000000058700000010262000000020540000000204400000002414000000000070000000011530000000000
300000000000000000140400000003633000000079500000001098500000003178000000016260000000075400000000280000000028300000011 78
800000008900000001463300000005854000000079500000000017710000000143300000000740000000068560000002913600000029000000000084
600000004608300000003646000000017620000000703400000001133000000045680000004183900000000071000000016100000002813300000001
625000000010130000000003800000001417000000423330000000143400000000020000000007000000013430000128233200000000208000000 0
007900000000033000000013900000000059000000061400000000048000000092000000004740000000254400000000840000028926800000 2
482430000000112000000012578000000005150000000037900000224709000000038850000002490500000230799000000316050000005132300000
026452000000136560000001344600000003900000002358400000001264000000059390000000117200000021040000000019700000001019000 0
000639100000004320000000053250000000430000000079800000000323000000009100000000287200000001809
010110          AUBURN, AL                 0000000003000000022767000000284710000000084000000000962000000138440000000 23
479000000045840000000023000000003240000000023200000028471000000012870000000355100000001411000000236330000000021700000000
448800000010324000000002320000000481500000003810000000007330000000311000000014965000000003120000000005000000000110000000
```

```
3
001040000000003700000000000000000000000000000000000000000000001649000000056960000000157600000000139000000000180000000010700000
006657000000108800000000649000000010875000000005510000001087700000004930000000037290000000893700000000319000000018960000
000198400000003143000000007220000001032400000004170000000003560000005520000000006154000000002020000000023100000000573000
000103240000000585300000004226000000000600000000122000000010271000000065260000000180700000000074000000092120000000239200
000009793000000080530000000468500000017667000000023680000000011360000000027300000000004000000138780000001121000000011097 0
000004891000000005150000000473200000000398000000105820000000087800000002088000000051010000000006000000000453000000000 0
300000000000000000041600000003015000000049110000001039700000002058000000012980000000064100000000418000000031900000008 70
700000000948000000192310000000547100000004911000000007920000000066600000000315000000080780000002397000000000982000000048
060000000749500000001128000000002870000000421800000000852000000015510000000691300000000567000000004960000000008100000001
051000000010960000000003000000000243000000129330000000454000000000200000000005000000003270000032067100000000234300000 0
000030000000000300000000003000000000030000000000300000000003000000000030000000000000000000000900000005805400000
000000000000003300000000395000000002160000000018000000111315000000021880000001261900000112683000000258120000002001400000
019320000000088760000000816400000000174000000076494000000000000000000060900000001110400000000000000000000000684000000000400 0
000000000000004320000000056100000000446000000007960000000033700000000906000000026110000000 1921
0101751000      BESSEMER, AL               0000000001500000033428000000317290000000074400000002101000000172700000001 5
414000000162680000000003600000000011000000002910000031729000000025820000000667200000004826000000224750000000030300000000
053100000011207000000002780000000830000000005881000000020780000000273300000028711000000006460000000016000000000204000000
004140000000013000000000001000000001690000000053300000003207000000100360000000287500000000024000000000240000000000000000
000907000000119000000001089000000011899000000069200000011900000000924900000001303000000076140000000011100000000062100000
000113600000002371000000029280000001120700000006501000000002780000002900000000004706000000001590000000016800000000874000
000112070000000982600000001240000000000000000000134900000010108000000073820000000201100000000298000000093170000000554400
000018574000000093430000001303000000803000000004720000000020030000000655000000031500000001099000000114760000000114520
000000499800000001087000000017210000000034100000010365000000028130000000222300000001876000000000300000000065300000000000
300000000000000000188900000003305000000083140000001119400000002974000000018960000000078700000002290000000022300000011 54
800000000906000000144480000005202000000083140000000205800000001726000000010270000000804100000031157000000031350000000 91
320000000961500000002217000000056400000004710000000159600000001835000000092570000000047300000000864000000004800000001
858000000009000000000050000000029200000028450000000090000000003000000000070000000039400000056214500000000155000000 0
004900000000018000000007260000000034000000047000000000280000000053000000003680000000226800000000440000006964740000 0
000000000000044700000004357000000003890000000027800000131267000000024720000001611500000136367000000309970000003737300000
022945000000084570000000941800000002400000001537100000001530000000023180000000098500000013976000000001270000000066000000
000420220000003428000000052150000000424000000008030000000033000000009060000000294300000001891
0101851000      BIRMINGHAM, AL             0000000009900000300910000028441300000000050000000028870000015420800000124
767000001582000000000220000000009330000000205400000284413000000212810000005447800000039399000002086540000000029700000000
431700000107238000000002610000007276900000050247000000019470000000306440000024120200000005056000000010050000000178000000
034180000000012000000000015000000047610000000167400000034249000000119720000003095700000001386000000006740000000064800000
031815000001145030000010537000000114463000000072250000011447100000076074000000276600000007997900000001874000000055350000
```

(continued on next page)

(continued from previous page)

```
000900300000019834000000317640000010723800000057158000000002750000003130000000050080000000001870000000106600000005877000
001072380000009097700000014808000000001420000000175600000111755000000774340000002200400000000745300000102606000000247490O
000168412000001016990000002182200000076640000000380010000001799200000004894000000001252000000183540000012562100000125450O
000005880300000010882000000212050000000364300000114568000000192680000000256070000000278540000000000030000000009300000134371
000000227510000001262900000035676000000073046000001073250000003129100000017390000000076140000000256900000002212000001195
100000000875000000152100000000583300000073046000000187770000000129340000000761600000061658000002800040000002336300000746
030000014997200000037111000000005486000000740080000001448400000023498000001005730000000007580000000793000000001188000000018
165000000096750000000005100000000354000003028280000000106500000000001000000000040000000341100004660697000000074810000000
046900000000205000000089250000000317000000424700000000227000000000439000000002724000000186480000000009070000288018600001 7
446140000000155000000018816700000000236700000000186100001271277000000024119000000166433000012926350000002823580000001969310000 0
169358000000106372000000764300000000220300000384912400000000000000000055741000000132070000037202700000000133000000001492100 00
01205810000000432000000005215000000000424000000000803000000000330000000000906000000002943000000001891
RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0

5           0101851000      BIRMINGHAM, AL               0000000009900000300910000002844130000000000500000000288

     101    70000015420800000124767000001582000000000002200000000093300000000205400000284413000000212810000005 4478
     201    00000039399000002086540000000029700000000431700000107238000000000261000000727690000000502470000000194700
     301    00000306440000024120200000000505600000000100500000000017800000000341800000000120000000000150000000047610 0
     401    00000167400000034249000000011972000000030957000000001386000000000674000000000648000000318150000000114503000
     501    0010537000000011446300000000722500000011447100000007607400000002766000000007997900000000187400000000553500 00
     601    000900300000019834000000317640000010723800000057158000000002750000003130000000050080000000000187000 00
     701    00106600000005877000001072380000000909770000000148080000000001420000000175600000111755000000077434000000
```

```
4
RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0

     801    22004000000074530000010260600000024749000001684120000010169900000002182200000076640000000380010000001

     901    79920000000489400000000125200000018354000001256210000012545000000058803000000108820000002120500000003
     1001   64300000114568000000192680000000256070000000278540000000000030000000009300000134371000000227510000000126
     1101   29000000356760000000730460000001073250000000312910000001739000000000761400000002569000000022120000001195
     1201   10000000087500000015210000000058330000000730460000000187770000000129340000000076160000000616580000280004
     1301   00000023363000000746030000014997200000003711100000000548600000007400800000001448400000023498000001005730
     1401   00000007580000000793000000001188000000181650000000096750000000000510000000035400000302828000000000106500
     1501   00000000100000000000400000003411000046606970000000074810000000004690000000002050000000089250000000000317000
     1601   00004247000000000227000000000439000000002724000000018648000000000907000028801860000017446140000000155000000
     1701   01881670000000236700000000186100001271277000000024119000000166433000012926350000002823580000001969310000 0
     1801   16935800000010637200000007643000000000220300000384912400000000000000000055741000000132070000037202700000 0
     1901   013300000000149210000012058100000000432000000005215000000000424000000000803000000000330000000000906000000 0
     2001   294300000001891
```

Using PROC UNIVARIATE The variables to be analyzed are continuous variables. For the tables generated by later SAS statements, the variables need to have only a few discrete values. PROC UNIVARIATE provides descriptive statistics that can help in determining appropriate cutoff points for recoding original continuous values into new categorical values.

```
PROC UNIVARIATE;
   VAR POP1980 OVER65 INCOME HOUSING ELECTRIC CRIME JANTEMP
       JULYTEMP RAINFALL;
   TITLE1 'CENSUS DATA: OUTPUT FROM PROC UNIVARIATE';
   TITLE2 'TO DETERMINE CUTOFFS';
   ID CITY;
```

Output 10.2 Census Data: Output from PROC UNIVARIATE to Determine Cutoffs

```
                                CENSUS DATA: OUTPUT FROM PROC UNIVARIATE                                1
                                        TO DETERMINE CUTOFFS

                                            UNIVARIATE

VARIABLE=POP1980        1980 POPULATION ESTIMATE


                  MOMENTS                                                    QUANTILES(DEF=4)

N                    418  SUM WGTS         418                  100% MAX    7071639     99%       1670509
MEAN              177580  SUM            74228554               75% Q3       149898     95%        557788
STD DEV           430825  VARIANCE     1.856E+11                50% MED     84589.5     90%        345633
SKEWNESS         11.4994  KURTOSIS       166.119                25% Q1      63361.3     10%       54335.3
USS           9.058E+13   CSS          7.740E+13                 0% MIN       50211      5%       52381.2
CV               242.608  STD MEAN       21072.3                                         1%       50390.5
T:MEAN=0         8.42718  PROB>|T|        0.0001                RANGE       7021428
SGN RANK         43785.5  PROB>|S|        0.0001                Q3-Q1        86536.8
NUM ^= 0             418                                       MODE          50211


                                            EXTREMES

                                  LOWEST    ID          HIGHEST    ID
                                   50211(WESTMINS)      1595138(HOUSTON,)
                                   50308(LA MESA,)      1688210(PHILADEL)
                                   50319(WAUKESHA)      2966850(LOS ANGE)
                                   50363(ENID, OK)      3005072(CHICAGO,)
                                   50508(IOWA CIT)      7071639(NEW YORK)
```

```
                                CENSUS DATA: OUTPUT FROM PROC UNIVARIATE                                2
                                        TO DETERMINE CUTOFFS

                                            UNIVARIATE

VARIABLE=OVER65         PERSONS 65 & OVER (1980)


                  MOMENTS                                                    QUANTILES(DEF=4)

N                    418  SUM WGTS         418                  100% MAX     951732     99%        218948
MEAN            20552.9  SUM            8591127                75% Q3      16521.5     95%       67444.4
STD DEV         54847.8  VARIANCE    3008280924                50% MED       10012     90%       40961.7
SKEWNESS        12.8835  KURTOSIS       204.721                25% Q1      6795.75     10%        4523.7
USS           1.431E+12  CSS          1.254E+12                 0% MIN        1062      5%       3398.85
CV               266.861  STD MEAN       2682.69                                         1%       1867.24
T:MEAN=0         7.66131  PROB>|T|        0.0001                RANGE        950670
SGN RANK         43785.5  PROB>|S|        0.0001                Q3-Q1        9725.75
NUM ^= 0             418                                       MODE           6500


                                            EXTREMES

                                  LOWEST    ID          HIGHEST    ID
                                    1062(SANDY CI)       140508(DETROIT,)
                                    1421(CERRITOS)       237370(PHILADEL)
                                    1635(WESTMINS)       314216(LOS ANGE)
                                    1849(EAST LAN)       342511(CHICAGO,)
                                    1945(PLANO, T)       951732(NEW YORK)
```

```
                                CENSUS DATA: OUTPUT FROM PROC UNIVARIATE                                3
                                        TO DETERMINE CUTOFFS

                                            UNIVARIATE

VARIABLE=INCOME         PER CAPITA MONEY INCOME (1979)


                  MOMENTS                                                    QUANTILES(DEF=4)

N                    418  SUM WGTS         418                  100% MAX      18082     99%       12661.2
MEAN            7569.34  SUM            3163985                75% Q3         8340     95%       10626.3
STD DEV         1627.25  VARIANCE       2647929                50% MED        7296     90%        9722.1
SKEWNESS        1.30392  KURTOSIS       4.43919                25% Q1      6465.25     10%        5860.9
USS           2.505E+10  CSS         1104186288                 0% MIN        3681      5%       5591.15
CV               21.4978  STD MEAN       79.5912                                         1%       4172.89
```

(continued on next page)

(continued from previous page)

T:MEAN=0	95.1028	PROB>|T|	0.0001	RANGE	14401
SGN RANK	43785.5	PROB>|S|	0.0001	Q3-Q1	1874.75
NUM -= 0	418			MODE	5819

EXTREMES

LOWEST	ID	HIGHEST	ID
3681	(EAST ST.)	12632	(FARMINGT)
3714	(LAREDO,)	12668	(SOUTHFIE)
3966	(CAMDEN,)	12675	(WALNUT C)
4129	(BROWNSVI)	12799	(PALO ALT)
4360	(COMPTON,)	18082	(NEWPORT)

CENSUS DATA: OUTPUT FROM PROC UNIVARIATE 4
TO DETERMINE CUTOFFS

UNIVARIATE

VARIABLE=HOUSING HOUSING UNITS (1980)

MOMENTS

N	418	SUM WGTS	418				
MEAN	71405.8	SUM	29847608				
STD DEV	177443	VARIANCE	3.149E+10				
SKEWNESS	11.7225	KURTOSIS	172.955				
USS	1.526E+13	CSS	1.313E+13				
CV	248.5	STD MEAN	8679.03				
T:MEAN=0	8.22739	PROB>|T|	0.0001				
SGN RANK	43785.5	PROB>|S|	0.0001				
NUM -= 0	418						

QUANTILES(DEF=4)

100% MAX	2946410	99%	684240
75% Q3	58983.8	95%	227987
50% MED	32810.5	90%	142381
25% Q1	25102.3	10%	21118.8
0% MIN	13119	5%	19284.4
		1%	14843.3
RANGE	2933291		
Q3-Q1	33881.5		
MODE	13119		

EXTREMES

LOWEST	ID	HIGHEST	ID
13119	(EAST LAN)	678324	(HOUSTON,)
13682	(SANDY CI)	685629	(PHILADEL)
14353	(BALDWIN)	1174706	(CHICAGO,)
14826	(OREM, UT)	1189475	(LOS ANGE)
14917	(CERRITOS)	2946410	(NEW YORK)

CENSUS DATA: OUTPUT FROM PROC UNIVARIATE 5
TO DETERMINE CUTOFFS

UNIVARIATE

VARIABLE=ELECTRIC TYPICAL RESID. ELECTRIC BILL/MO. (1982)

MOMENTS

N	418	SUM WGTS	418				
MEAN	47.8885	SUM	20017.4				
STD DEV	11.7452	VARIANCE	137.951				
SKEWNESS	0.429251	KURTOSIS	1.72847				
USS	1016127	CSS	57525.4				
CV	24.5262	STD MEAN	0.574478				
T:MEAN=0	83.3599	PROB>|T|	0.0001				
SGN RANK	43785.5	PROB>|S|	0.0001				
NUM -= 0	418						

QUANTILES(DEF=4)

100% MAX	90.7	99%	89.38
75% Q3	54.57	95%	65.9934
50% MED	48.63	90%	62.4359
25% Q1	39.0825	10%	34.268
0% MIN	10.34	5%	30.061
		1%	18.7422
RANGE	80.36		
Q3-Q1	15.4875		
MODE	54.57		

EXTREMES

LOWEST	ID	HIGHEST	ID
10.34	(SEATTLE,)	89.38	(MOUNT VE)
16.15	(TACOMA,)	89.38	(NEW ROCH)
16.82	(PALO ALT)	89.38	(YONKERS,)
18.29	(SPOKANE,)	90.53	(NEW YORK)
20.67	(BELLEVUE)	90.7	(HONOLULU)

CENSUS DATA: OUTPUT FROM PROC UNIVARIATE 6
TO DETERMINE CUTOFFS

UNIVARIATE

VARIABLE=CRIME CRIME RATE/100,000 POP (1981)

MOMENTS

N	418	SUM WGTS	418
MEAN	7856.02	SUM	3283815
STD DEV	2774.58	VARIANCE	7698271
SKEWNESS	0.766309	KURTOSIS	1.27228
USS	2.901E+10	CSS	3210178999
CV	35.3178	STD MEAN	135.709
T:MEAN=0	57.8887	PROB>|T|	0.0001
SGN RANK	43785.5	PROB>|S|	0.0001
NUM ^= 0	418		

QUANTILES(DEF=4)

100% MAX	20900	99%	15923.9
75% Q3	9428.75	95%	12848.7
50% MED	7473.5	90%	11544.9
25% Q1	5911	10%	4760.5
0% MIN	1066	5%	4034.35
		1%	2449.12
RANGE	19834		
Q3-Q1	3517.75		
MODE	4100		

EXTREMES

LOWEST	ID	HIGHEST	ID
1066	(WALTHAM,)	15715	(CAMDEN,)
1824	(LOWELL,)	15973	(DAYTONA)
2306	(LAKEWOOD)	17131	(WEST PAL)
2402	(ALTOONA,)	17930	(HARTFORD)
2650	(LORAIN,)	20900	(RACINE,)

CENSUS DATA: OUTPUT FROM PROC UNIVARIATE 7
TO DETERMINE CUTOFFS

UNIVARIATE

VARIABLE=JANTEMP MEAN JANUARY TEMPERATURE (DEGREES F)

MOMENTS

N	418	SUM WGTS	418
MEAN	36.6038	SUM	15300.4
STD DEV	13.7468	VARIANCE	188.976
SKEWNESS	0.230499	KURTOSIS	-1.00507
USS	638856	CSS	78802.8
CV	37.5557	STD MEAN	0.67238
T:MEAN=0	54.4392	PROB>|T|	0.0001
SGN RANK	43785.5	PROB>|S|	0.0001
NUM ^= 0	418		

QUANTILES(DEF=4)

100% MAX	72.6	99%	66.6859
75% Q3	49.2	95%	57.2
50% MED	32.35	90%	55.1
25% Q1	25.5	10%	21.08
0% MIN	4.3	5%	18.695
		1%	10.876
RANGE	68.3		
Q3-Q1	23.7		
MODE	29.6		

EXTREMES

LOWEST	ID	HIGHEST	ID
4.3	(FARGO, N)	66.2	(HOLLYWOO)
6.3	(DULUTH,)	66.8	(POMPANO)
9.8	(EAU CLAI)	67.1	(MIAMI, F)
10.8	(ROCHESTE)	68	(MIAMI BE)
11.2	(ST. PAUL)	72.6	(HONOLULU)

CENSUS DATA: OUTPUT FROM PROC UNIVARIATE 8
TO DETERMINE CUTOFFS

UNIVARIATE

VARIABLE=JULYTEMP MEAN JULY TEMPERATURE (DEGREES F)

MOMENTS

N	418	SUM WGTS	418
MEAN	75.139	SUM	31408.1
STD DEV	5.79746	VARIANCE	33.6105
SKEWNESS	0.191604	KURTOSIS	0.0417272
USS	2373989	CSS	14015.6
CV	7.71565	STD MEAN	0.283563
T:MEAN=0	264.982	PROB>|T|	0.0001
SGN RANK	43785.5	PROB>|S|	0.0001
NUM ^= 0	418		

QUANTILES(DEF=4)

100% MAX	92.3	99%	90
75% Q3	78.9	95%	84.71
50% MED	74.05	90%	82.71
25% Q1	71.675	10%	68.37
0% MIN	58.1	5%	65.7
		1%	61.7
RANGE	34.2		
Q3-Q1	7.225		
MODE	73.5		

(continued on next page)

(continued from previous page)

```
                                EXTREMES

                LOWEST    ID          HIGHEST    ID
                  58.1(ANCHORAG)          90(MESA, AZ)
                  59.5(SALINAS,)          90(SCOTTSDA)
                  61.7(WALNUT C)        90.3(LAS VEGA)
                  61.7(CONCORD,)        92.3(GLENDALE)
                  61.7(BERKELEY)        92.3(PHOENIX,)
```

```
                                CENSUS DATA: OUTPUT FROM PROC UNIVARIATE                    9
                                      TO DETERMINE CUTOFFS

                                          UNIVARIATE

VARIABLE=RAINFALL       ANNUAL PRECIPITATION


                  MOMENTS                                                 QUANTILES(DEF=4)

N                      418  SUM WGTS          418                  100% MAX      64.64      99%       61.3381
MEAN              32.8366  SUM            13725.7                   75% Q3       43.78      95%         52.79
STD DEV           13.4742  VARIANCE       181.555                   50% MED      34.71      90%         48.463
SKEWNESS        -0.158253  KURTOSIS      -0.82774                   25% Q1       19.29      10%         13.699
USS                526414  CSS            75708.5                    0% MIN       4.19       5%         11.54
CV                41.0342  STD MEAN      0.659047                                           1%         7.1822
T:MEAN=0          49.8244  PROB>|T|        0.0001                   RANGE        60.45
SGN RANK          43785.5  PROB>|S|        0.0001                   Q3-Q1        24.49
NUM ^= 0              418                                         MODE         30.97


                                EXTREMES

                LOWEST    ID          HIGHEST    ID
                  4.19(LAS VEGA)          61.16(PENSACOL)
                  5.72(BAKERSFI)          61.38(POMPANO )
                  7.11(PHOENIX,)          63.47(HIALEAH,)
                  7.11(GLENDALE)          64.59(TALLAHAS)
                  7.49(RENO, NV)          64.64(MOBILE, )
```

Data set CITIES Based on the UNIVARIATE results, the programming statements in this DATA step define the categories. A LABEL statement provides labels for the new variables.

Also in this DATA step, two new geographic variables are created to represent the official U.S. region (variable REGION) and division (variable DIVISION) for each city. The original data contain a two-digit numeric code representing the state, along with the name of the city represented as a character variable. The FIPSTATE function used with the STATE variable (the numeric code) obtains the two-digit character abbreviation for each state. Then the INDEX function checks for values of ST (state abbreviation) and SAS assigns states to the DIVISION variable. IF-THEN/ELSE statements code the nine geographical divisions into the four U.S. regions of the country.

```
DATA CITIES;
   LENGTH REGION $13 ST $3;
   SET CCDB83;

   IF OVER65<=7000 THEN OVR65GRP=1;
   ELSE IF 7000<OVER65<=17000 THEN OVR65GRP=2;
   ELSE IF OVER65>17000 THEN OVR65GRP=3;

   IF INCOME<=7569 THEN INCOMGRP=1;
   ELSE IF INCOME>7569 THEN INCOMGRP=2;

   IF HOUSING<=25000 THEN HOUSGRP=1;
   ELSE IF 25000<HOUSING<=60000 THEN HOUSGRP=2;
   ELSE IF HOUSING>60000 THEN HOUSGRP=3;
```

```
IF ELECTRIC<=40.00 THEN ELECGRP=1;
ELSE IF 40<ELECTRIC<=55.00 THEN ELECGRP=2;
ELSE IF ELECTRIC>55 THEN ELECGRP=3;

IF CRIME<=6000 THEN CRIMEGRP=1;
ELSE IF 6000<CRIME<=7500 THEN CRIMEGRP=2;
ELSE IF 7500<CRIME<=9500 THEN CRIMEGRP=3;
ELSE IF CRIME>9500 THEN CRIMEGRP=4;

IF JANTEMP<=32.0 THEN JANGRP=1;
ELSE IF JANTEMP>32 THEN JANGRP=2;

IF JULYTEMP<=75.0 THEN JULYGRP=1;
ELSE IF JULYTEMP>75.0 THEN JULYGRP=2;

IF RAINFALL<19 THEN RAINGRP=1;
ELSE IF 19<=RAINFALL<=44 THEN RAINGRP=2;
ELSE IF RAINFALL>44 THEN RAINGRP=3;

LABEL POPGRP='1980 POPULATION ESTIMATE'
      OVR65GRP='PERSONS 65 & OVER (1980)'
      INCOMGRP='PER CAPITA MONEY INCOME (1979)'
      HOUSGRP='HOUSING UNITS (1980)'
      ELECGRP='TYPICAL RESID. ELECTRIC BILL/MO. (1982)'
      CRIMEGRP='CRIME RATE/100,000 POP (1981)'
      JANGRP='MEAN JANUARY TEMPERATURE (DEGREES F)'
      JULYGRP='MEAN JULY TEMPERATURE (DEGREES F)'
      RAINGRP='ANNUAL PRECIPITATION (INCHES)';

ST=FIPSTATE(STATE);

IF INDEX('MS AL TN KY ',ST)¬=0
   THEN DIVISION='EAST SOUTH CENTRAL';
IF INDEX('FL GA SC NC VA WV MD DE DC ',ST)¬=0
   THEN DIVISION='SOUTH ATLANTIC';
IF INDEX('AR LA OK TX ',ST)¬=0
   THEN DIVISION='WEST SOUTH CENTRAL';
IF INDEX('PA NJ NY ',ST)¬=0
   THEN DIVISION='MIDDLE ATLANTIC';
IF INDEX('VT NH ME MA CT RI ',ST)¬=0
   THEN DIVISION='NEW ENGLAND';
IF INDEX('IL IN OH MI WI ',ST)¬=0
   THEN DIVISION='EAST NORTH CENTRAL';
IF INDEX('MN IA MO KS NE SD ND ',ST)¬=0
   THEN DIVISION='WEST NORTH CENTRAL';
IF INDEX('MT WY CO UT ID NV AZ NM ',ST)¬=0
   THEN DIVISION='MOUNTAIN';
IF INDEX('WA OR CA AK HI ',ST)¬=0
   THEN DIVISION='PACIFIC';

IF DIVISION='EAST SOUTH CENTRAL' OR
   DIVISION='SOUTH ATLANTIC' OR
   DIVISION='WEST SOUTH CENTRAL' THEN
   REGION='SOUTH';
```

```
ELSE IF DIVISION='MIDDLE ATLANTIC' OR
   DIVISION='NEW ENGLAND' THEN
   REGION='NORTHEAST';
ELSE IF DIVISION='EAST NORTH CENTRAL' OR
   DIVISION='WEST NORTH CENTRAL' THEN
   REGION='NORTH CENTRAL';
ELSE IF DIVISION='MOUNTAIN' OR
   DIVISION='PACIFIC' THEN
   REGION='WEST';

IF CITYGOVT=0 THEN CITYGOVT=.;
```

Grouping values with PROC FORMAT The new variables in the data set CITIES are now categorical, but their numerical values are not very descriptive for the tables that are planned. To obtain more easily understood values, we use PROC FORMAT to define formats for these variables. FORMAT statements in later PROC steps will associate these formats with the appropriate variables.

```
PROC FORMAT;
   VALUE PC 1='50,001-75,000'
            2='75,001-100,000'
            3='OVER 100,000';

   VALUE OC 1=7000 OR LESS
            2='7001-17000'
            3=OVER 17000;

   VALUE IC 1=AVERAGE OR ABOVE
            2=BELOW AVERAGE;

   VALUE HC 1=LOW
            2=MEDIUM
            3=HIGH;

   VALUE EC 1=LOW
            2=MEDIUM
            3=HIGH;

   VALUE GC 1=MAYOR COUNCIL
            2=COUNCIL MANAGER
            3=COMMISSION;

   VALUE CC 1=6000 OR LESS
            2='6001-7500'
            3='7501-9500'
            4=OVER 9500;

   VALUE JAN 1=UNDER 32 DEGREES
             2=32 DEGREES OR HIGHER;

   VALUE JUL 1=UNDER 75 DEGREES
             2=75 DEGREES OR HIGHER;
```

```
VALUE RC 1=LESS THAN 19 INCHES
         2='19-44 INCHES'
         3=OVER 44 INCHES;
```

Printing the values The PRINT procedure prints the unformatted values of the variables in the data set CITIES. (Although the formats could have been used with PROC PRINT, using unformatted values here results in a more concise printout.)

```
PROC PRINT;
   VAR CITY ST REGION DIVISION POP1980 OVER65 INCOME HOUSING
      ELECTRIC CITYGOVT CRIME JANTEMP JULYTEMP RAINFALL
      POPGRP OVR65GRP INCOMGRP HOUSGRP ELECGRP CRIMEGRP JANGRP
      JULYGRP RAINGRP;
   TITLE 'CENSUS DATA: OUTPUT FROM PRINT PROCEDURE';
   TITLE2;
RUN;
```

Output 10.3 Census Data: Output from Print Procedure

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 10

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT	CRIME
1	BIRMINGHAM, AL	AL	SOUTH	EAST SOUTH CENTRAL	284413	39399	5833	114503	43.20	1	11972
2	HUNTSVILLE, AL	AL	SOUTH	EAST SOUTH CENTRAL	142513	10039	7661	53281	31.57	1	8370
3	MOBILE, AL	AL	SOUTH	EAST SOUTH CENTRAL	200452	22311	6593	75577	43.20	3	10878
4	MONTGOMERY, AL	AL	SOUTH	EAST SOUTH CENTRAL	177857	17916	6640	67417	43.20	1	6418
5	TUSCALOOSA, AL	AL	SOUTH	EAST SOUTH CENTRAL	75211	7341	5633	28200	43.20	3	7655
6	ANCHORAGE, AK	AK	WEST	PACIFIC	174431	3520	11339	70363	38.90	1	7665
7	GLENDALE, AZ	AZ	WEST	MOUNTAIN	97172	6796	7091	35458	50.46	2	5923
8	MESA, AZ	AZ	WEST	MOUNTAIN	152453	17067	7196	65299	50.46	2	7256
9	PHOENIX, AZ	AZ	WEST	MOUNTAIN	789704	73493	7551	308302	56.19	2	10235
10	SCOTTSDALE, AZ	AZ	WEST	MOUNTAIN	88412	10855	10342	42041	50.46	2	8470
11	TEMPE, AZ	AZ	WEST	MOUNTAIN	106743	5008	7910	40206	50.46	2	10051
12	TUCSON, AZ	AZ	WEST	MOUNTAIN	330537	38523	6473	137249	46.45	2	10922
13	FORT SMITH, AR	AR	SOUTH	WEST SOUTH CENTRAL	71626	9474	7262	30385	35.72	2	8390
14	LITTLE ROCK, AR	AR	SOUTH	WEST SOUTH CENTRAL	158461	17369	8002	64674	38.26	2	12156
15	NORTH LITTLE ROCK, AR	AR	SOUTH	WEST SOUTH CENTRAL	64288	7915	6980	25914	41.98	1	7764
16	PINE BLUFF, AR	AR	SOUTH	WEST SOUTH CENTRAL	56636	7891	5575	21300	38.26	1	6798
17	ALAMEDA, CA	CA	WEST	PACIFIC	63852	7709	9288	27802	53.03	2	6970
18	ALHAMBRA, CA	CA	WEST	PACIFIC	64615	10241	7772	27193	54.57	2	6310
19	ANAHEIM, CA	CA	WEST	PACIFIC	219311	16987	8532	82725	57.77	2	7670
20	BAKERSFIELD, CA	CA	WEST	PACIFIC	105611	9726	8219	42761	53.18	2	11846
21	BALDWIN PARK, CA	CA	WEST	PACIFIC	50554	3141	4955	14353	54.57	2	6480
22	BELLFLOWER, CA	CA	WEST	PACIFIC	53441	5845	7694	22263	54.57	2	7041
23	BERKELEY, CA	CA	WEST	PACIFIC	103328	11132	8461	46334	53.18	2	13754
24	BUENA PARK, CA	CA	WEST	PACIFIC	64165	3604	8261	21990	54.57	2	6237
25	BURBANK, CA	CA	WEST	PACIFIC	84625	12764	9251	37127	62.12	2	5456

OBS	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
1	42.4	80.3	52.15	3	3	1	3	2	4	2	2	3
2	40.2	79.3	54.74	3	2	2	2	1	3	2	2	3
3	50.8	82.2	64.64	3	3	1	3	2	4	2	2	3
4	46.7	81.7	49.16	3	3	1	3	2	2	2	2	3
5	43.8	81.3	52.61	2	2	1	2	2	3	2	2	3
6	13.0	58.1	15.20	3	1	2	3	1	3	1	1	1
7	52.3	92.3	7.11	2	1	1	2	2	1	2	2	1
8	51.1	90.0	7.85	3	3	1	3	2	2	2	2	1
9	52.3	92.3	7.11	3	3	1	3	3	4	2	2	1
10	51.1	90.0	7.85	2	2	2	2	2	3	2	2	1
11	50.9	89.7	8.00	3	1	2	2	2	4	2	2	1
12	51.1	86.2	11.14	3	3	1	3	2	4	2	2	1
13	37.5	82.1	39.91	1	2	1	2	1	3	2	2	2
14	39.9	82.1	49.20	3	3	2	3	1	4	2	2	3
15	39.5	81.7	46.26	1	2	1	2	2	3	2	2	3
16	42.9	82.7	50.28	1	2	1	1	1	2	2	2	3
17	49.0	63.7	18.03	1	2	2	2	2	2	2	1	1
18	54.8	74.8	17.76	1	2	2	2	2	2	2	1	1
19	55.9	72.2	12.60	3	2	2	3	3	3	2	1	1
20	48.2	84.5	5.72	3	2	2	2	2	4	2	2	1
21	54.8	74.8	17.76	1	1	1	1	2	2	2	1	1

(continued on next page)

(continued from previous page)

22	57.2	74.1	14.85	1	1	2	1	2	2	2	1	1
23	49.7	61.7	23.24	3	2	2	2	2	4	2	1	2
24	54.9	73.5	14.46	1	1	2	1	2	2	2	1	1
25	53.8	75.1	15.78	2	2	2	2	3	1	2	2	1

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 11

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT	CRIME
26	CARSON, CA	CA	WEST	PACIFIC	81221	4388	7177	23259	54.57	2	6632
27	CERRITOS, CA	CA	WEST	PACIFIC	53020	1421	9003	14917	54.57	2	7777
28	CHULA VISTA, CA	CA	WEST	PACIFIC	83927	8609	7464	31888	79.05	2	6330
29	COMPTON, CA	CA	WEST	PACIFIC	81286	3275	4360	22427	54.57	2	10810
30	CONCORD, CA	CA	WEST	PACIFIC	103255	7568	8880	39490	53.18	2	7310
31	COSTA MESA, CA	CA	WEST	PACIFIC	82562	6440	8985	34023	54.57	2	8580
32	DALY CITY, CA	CA	WEST	PACIFIC	78519	6850	8227	27823	53.18	2	4409
33	DOWNEY, CA	CA	WEST	PACIFIC	82602	9130	9339	33688	54.57	2	5277
34	EL CAJON, CA	CA	WEST	PACIFIC	73892	7954	7166	30097	79.05	2	7959
35	EL MONTE, CA	CA	WEST	PACIFIC	79494	6355	5002	25393	54.57	2	7965
36	ESCONDIDO, CA	CA	WEST	PACIFIC	64355	10471	7151	27153	79.05	2	5716
37	FAIRFIELD, CA	CA	WEST	PACIFIC	58099	2546	6778	18951	53.18	2	6682
38	FOUNTAIN VALLEY, CA	CA	WEST	PACIFIC	55080	2263	9423	16758	54.57	2	4931
39	FREMONT, CA	CA	WEST	PACIFIC	131945	6880	9087	45486	53.18	2	5914
40	FRESNO, CA	CA	WEST	PACIFIC	218202	23810	6733	88749	53.18	2	11116
41	FULLERTON, CA	CA	WEST	PACIFIC	102034	8112	10156	39506	54.57	2	6383
42	GARDEN GROVE, CA	CA	WEST	PACIFIC	123307	9073	8027	42846	54.57	2	7449
43	GLENDALE, CA	CA	WEST	PACIFIC	139060	22687	9514	61653	62.85	2	5449
44	HAWTHORNE, CA	CA	WEST	PACIFIC	56447	4528	8039	23907	54.57	2	9397
45	HAYWARD, CA	CA	WEST	PACIFIC	94167	8325	8070	35870	53.18	2	9455
46	HUNTINGTON BEACH, CA	CA	WEST	PACIFIC	170505	10019	9782	63686	54.57	2	5350
47	INGLEWOOD, CA	CA	WEST	PACIFIC	94245	7860	6962	38235	54.57	2	11255
48	IRVINE, CA	CA	WEST	PACIFIC	62134	2285	12169	22514	54.57	2	4567
49	LAKEWOOD, CA	CA	WEST	PACIFIC	74654	5639	8504	26250	54.57	2	5708
50	LA MESA, CA	CA	WEST	PACIFIC	50308	7413	8761	22616	79.05	2	5207

OBS	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
26	55.1	69.1	13.13	2	1	1	1	2	2	2	1	1
27	56.0	69.0	12.08	1	1	2	1	2	3	2	1	1
28	53.4	66.9	8.67	2	2	1	2	3	2	2	1	1
29	55.1	69.1	13.13	2	1	1	1	2	4	2	1	1
30	49.7	61.7	23.24	3	2	2	2	2	2	2	1	2
31	54.2	66.7	11.09	2	1	2	2	2	3	2	1	1
32	48.5	62.2	19.71	2	1	2	2	2	1	2	1	2
33	56.0	69.0	12.08	2	2	2	2	2	1	2	1	1
34	55.4	71.8	12.56	1	2	1	2	3	3	2	1	1
35	54.8	74.8	17.76	2	1	1	2	2	3	2	1	1
36	52.2	73.3	14.53	1	2	1	2	3	1	2	1	1
37	46.3	71.7	20.94	1	1	1	1	2	2	2	1	2
38	55.9	72.2	12.60	1	1	2	1	2	1	2	1	1
39	47.8	66.3	14.77	3	1	2	2	2	1	2	1	1
40	45.5	81.0	10.52	3	3	1	3	2	4	2	2	1
41	54.9	73.5	14.46	3	2	2	2	2	2	2	1	1
42	55.9	72.2	12.60	3	2	2	2	2	2	2	1	1
43	55.0	74.7	19.29	3	3	2	3	3	1	2	1	2
44	55.1	69.1	13.13	1	1	2	1	2	3	2	1	1
45	45.9	71.3	14.11	2	2	2	2	2	3	2	1	1
46	54.2	66.7	11.09	3	2	2	3	2	1	2	1	1
47	56.5	69.2	14.09	2	2	1	2	2	4	2	1	1
48	55.9	72.2	12.60	1	1	2	1	2	1	2	1	1
49	55.2	72.8	11.54	1	1	2	2	2	1	2	1	1
50	55.4	71.8	12.56	1	2	2	1	3	1	2	1	1

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 12

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT	CRIME
51	LONG BEACH, CA	CA	WEST	PACIFIC	361334	50674	8343	159715	54.57	2	8810
52	LOS ANGELES, CA	CA	WEST	PACIFIC	2966850	314216	8408	1189475	51.97	1	10033
53	MODESTO, CA	CA	WEST	PACIFIC	106602	10329	7735	42391	27.09	2	9797
54	MONTEBELLO, CA	CA	WEST	PACIFIC	52929	5690	7153	18538	54.57	2	6334
55	MONTEREY PARK, CA	CA	WEST	PACIFIC	54338	5115	8019	19331	54.57	2	5936
56	MOUNTAIN VIEW, CA	CA	WEST	PACIFIC	58655	5400	10754	28576	53.18	2	7281
57	NAPA, CA	CA	WEST	PACIFIC	50879	6391	8377	20227	53.18	2	6872
58	NEWPORT BEACH, CA	CA	WEST	PACIFIC	62556	7288	18082	31397	54.57	2	8887
59	NORWALK, CA	CA	WEST	PACIFIC	85286	5225	6276	25827	54.57	2	5622
60	OAKLAND, CA	CA	WEST	PACIFIC	339337	44795	7701	150274	53.18	2	12848
61	OCEANSIDE, CA	CA	WEST	PACIFIC	76698	10510	7040	32733	79.05	2	7286
62	ONTARIO, CA	CA	WEST	PACIFIC	88820	6500	6772	31339	54.57	2	8502

(continued on next page)

(continued from previous page)

63	ORANGE, CA	CA	WEST	PACIFIC	91788	7202	8889	32888	54.57	2	6115
64	OXNARD, CA	CA	WEST	PACIFIC	108195	7142	6393	35087	54.57	2	6349
65	PALO ALTO, CA	CA	WEST	PACIFIC	55225	7408	12799	23747	16.82	2	8725
66	PASADENA, CA	CA	WEST	PACIFIC	118550	17686	9189	49732	47.78	2	9343
67	PICO RIVERA, CA	CA	WEST	PACIFIC	53459	3708	5878	15884	54.57	2	5878
68	POMONA, CA	CA	WEST	PACIFIC	92742	8280	5819	32153	54.57	2	11835
69	REDONDO BEACH, CA	CA	WEST	PACIFIC	57102	3825	10569	25867	54.57	2	6541
70	REDWOOD CITY, CA	CA	WEST	PACIFIC	54951	6253	9720	23491	53.18	2	6201
71	RICHMOND, CA	CA	WEST	PACIFIC	74676	7836	6977	29082	53.18	2	10696
72	SACRAMENTO, CA	CA	WEST	PACIFIC	275741	37484	7558	123284	21.88	2	13007
73	SALINAS, CA	CA	WEST	PACIFIC	80479	7338	6795	28383	53.18	2	8055
74	SAN BERNARDINO, CA	CA	WEST	PACIFIC	117490	14034	6411	46458	54.57	1	14444
75	SAN BUENAVENTURA %VENTURA<, CA	CA	WEST	PACIFIC	74393	9141	8644	30627	54.57	2	6478

OBS	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
51	55.2	72.8	11.54	3	3	2	3	2	3	2	1	1
52	57.2	74.1	14.85	3	3	2	3	2	4	2	1	1
53	45.7	76.8	11.70	3	2	2	2	1	4	2	2	1
54	54.8	74.8	17.76	1	1	1	1	2	2	2	1	1
55	54.8	74.8	17.76	1	1	2	1	2	1	2	1	1
56	48.7	68.4	19.27	1	1	2	2	2	2	2	1	2
57	47.6	67.7	24.34	1	1	2	1	2	2	2	1	2
58	54.2	66.7	11.09	1	2	2	2	2	3	2	1	1
59	54.9	73.5	14.46	2	1	1	2	2	1	2	1	1
60	49.0	63.7	18.03	3	3	2	3	2	4	2	1	1
61	52.2	73.3	14.53	2	2	1	2	3	2	2	1	1
62	52.4	74.6	17.02	2	1	1	2	2	3	2	1	1
63	55.9	72.2	12.60	2	2	2	2	2	2	2	1	1
64	54.6	65.7	14.53	3	2	1	2	2	2	2	1	1
65	47.8	66.3	14.77	1	2	2	1	1	3	2	1	1
66	55.0	74.7	19.29	3	3	2	2	2	3	2	1	2
67	57.2	74.1	14.85	1	1	1	1	2	1	2	1	1
68	52.4	74.6	17.02	2	2	1	2	2	4	2	1	1
69	55.1	69.1	13.13	1	1	2	2	2	2	2	1	1
70	48.7	68.4	19.27	1	1	2	1	2	2	2	1	2
71	49.7	62.4	21.83	1	2	1	2	2	4	2	1	2
72	47.1	76.6	17.87	3	3	1	3	1	4	2	2	1
73	51.4	59.5	18.35	2	2	1	2	2	3	2	1	1
74	52.9	79.3	15.68	3	2	1	2	2	4	2	2	1
75	54.6	65.7	14.53	1	2	2	2	2	2	2	1	1

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 13

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT	CRIME
76	SAN DIEGO, CA	CA	WEST	PACIFIC	875538	85313	8016	341928	80.55	2	7362
77	SAN FRANCISCO, CA	CA	WEST	PACIFIC	678974	104285	9265	316608	53.18	1	10620
78	SAN JOSE, CA	CA	WEST	PACIFIC	629442	39128	8379	216653	53.18	2	8454
79	SAN LEANDRO, CA	CA	WEST	PACIFIC	63952	10744	9453	28086	53.18	2	8284
80	SAN MATEO, CA	CA	WEST	PACIFIC	77561	11376	11074	34268	53.18	2	7489
81	SANTA ANA, CA	CA	WEST	PACIFIC	203713	15072	6569	67180	54.57	2	9807
82	SANTA BARBARA, CA	CA	WEST	PACIFIC	74414	13654	9103	33925	54.57	2	9783
83	SANTA CLARA, CA	CA	WEST	PACIFIC	87746	7128	9356	34858	34.25	2	7049
84	SANTA MONICA, CA	CA	WEST	PACIFIC	88314	14478	11126	46418	54.57	2	11531
85	SANTA ROSA, CA	CA	WEST	PACIFIC	83320	13184	8471	35192	53.18	2	7429
86	SIMI VALLEY, CA	CA	WEST	PACIFIC	77500	2690	7801	22643	54.57	2	3715
87	SOUTH GATE, CA	CA	WEST	PACIFIC	66784	7181	5734	23589	54.57	2	5854
88	STOCKTON, CA	CA	WEST	PACIFIC	149779	16514	6747	61315	53.18	2	11245
89	SUNNYVALE, CA	CA	WEST	PACIFIC	106618	8777	10359	44021	53.18	2	5803
90	THOUSAND OAKS, CA	CA	WEST	PACIFIC	77072	4686	9962	27491	54.57	2	4153
91	TORRANCE, CA	CA	WEST	PACIFIC	129881	11050	10285	50994	54.57	2	5410
92	VALLEJO, CA	CA	WEST	PACIFIC	80303	8734	7226	30319	53.18	2	7464
93	WALNUT CREEK, CA	CA	WEST	PACIFIC	53643	10834	12675	24405	53.18	2	5598
94	WEST COVINA, CA	CA	WEST	PACIFIC	80291	4475	8856	27375	54.57	2	8131
95	WESTMINSTER, CA	CA	WEST	PACIFIC	71133	5275	8454	24563	54.57	2	6648
96	WHITTIER, CA	CA	WEST	PACIFIC	69717	9180	9698	27796	54.57	2	4945
97	ARVADA, CO	CO	WEST	MOUNTAIN	84576	3877	8754	29360	48.84	2	5193
98	AURORA, CO	CO	WEST	MOUNTAIN	158588	6795	8925	62821	48.84	2	7307
99	BOULDER, CO	CO	WEST	MOUNTAIN	76685	5425	8385	30287	48.84	2	7614
100	COLORADO SPRINGS, CO	CO	WEST	MOUNTAIN	215150	17787	7404	88283	26.38	2	8841

OBS	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
76	56.8	70.3	9.32	3	3	2	3	3	2	2	1	1
77	48.5	62.2	19.71	3	3	2	3	2	4	2	1	2
78	49.5	68.8	13.86	3	3	2	3	2	3	2	1	1
79	49.0	63.7	18.03	1	2	2	2	2	3	2	1	1
80	48.7	68.4	19.27	2	2	2	2	2	2	2	1	2
81	55.9	72.2	12.60	3	2	1	3	2	4	2	1	1
82	54.0	66.6	17.70	1	2	2	2	2	4	2	1	1
83	49.5	68.8	13.86	2	2	2	2	1	2	2	1	1
84	56.4	65.8	13.69	2	2	2	2	2	4	2	1	1

(continued on next page)

(continued from previous page)

85	46.7	67.4	29.88	2	2	2	2	2	2	2	1	2
86	54.6	65.7	14.53	2	1	2	1	2	1	2	1	1
87	57.2	74.1	14.85	1	2	1	1	2	1	2	1	1
88	45.2	78.0	13.77	3	2	1	3	2	4	2	2	1
89	49.5	68.8	13.86	3	2	2	2	2	1	2	1	1
90	54.6	65.7	14.53	2	1	2	2	2	1	2	1	1
91	55.1	69.1	13.13	3	2	2	2	2	1	2	1	1
92	47.1	67.0	24.02	2	2	1	2	2	2	2	1	2
93	49.7	61.7	23.24	1	2	2	1	2	1	2	1	2
94	52.2	74.4	17.13	2	1	2	2	2	3	2	1	1
95	55.2	72.8	11.54	1	1	2	1	2	2	2	1	1
96	57.2	74.1	14.85	1	2	2	2	2	1	2	1	1
97	30.9	72.6	15.15	2	1	2	2	2	1	1	1	1
98	29.5	73.3	15.31	3	1	2	3	2	2	1	1	1
99	32.6	74.0	18.12	2	1	2	2	2	3	2	1	1
100	28.8	71.2	15.42	3	3	1	3	1	3	1	1	1

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 14

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
101	DENVER, CO	CO	WEST	MOUNTAIN	492365	61923	8555	227879	48.84	1
102	FORT COLLINS, CO	CO	WEST	MOUNTAIN	65092	4778	7130	25382	44.24	2
103	GREELEY, CO	CO	WEST	MOUNTAIN	53006	5172	6760	20731	43.61	2
104	LAKEWOOD, CO	CO	WEST	MOUNTAIN	112860	8093	9741	43418	48.84	2
105	PUEBLO, CO	CO	WEST	MOUNTAIN	101686	12515	6696	40012	33.11	2
106	WESTMINSTER, CO	CO	WEST	MOUNTAIN	50211	1635	8075	18560	48.84	2
107	BRIDGEPORT, CT	CT	NORTHEAST	NEW ENGLAND	142546	19052	6081	55291	72.37	1
108	BRISTOL, CT	CT	NORTHEAST	NEW ENGLAND	57370	6255	7721	21004	65.98	1
109	DANBURY, CT	CT	NORTHEAST	NEW ENGLAND	60470	6499	7957	22581	65.98	1
110	HARTFORD, CT	CT	NORTHEAST	NEW ENGLAND	136392	15499	5559	55254	61.93	2
111	MERIDEN, CT	CT	NORTHEAST	NEW ENGLAND	57118	7233	7496	22198	65.98	2
112	NEW BRITAIN, CT	CT	NORTHEAST	NEW ENGLAND	73840	10633	7156	29762	65.98	1
113	NEW HAVEN, CT	CT	NORTHEAST	NEW ENGLAND	126109	16544	5822	50634	72.37	1
114	NORWALK, CT	CT	NORTHEAST	NEW ENGLAND	77767	8423	9482	29448	65.98	1
115	STAMFORD, CT	CT	NORTHEAST	NEW ENGLAND	102453	12312	10711	40059	61.93	1
116	WATERBURY, CT	CT	NORTHEAST	NEW ENGLAND	103266	15957	6429	40854	65.98	1
117	WEST HAVEN, CT	CT	NORTHEAST	NEW ENGLAND	53184	6860	7200	20915	72.37	1
118	WILMINGTON, DE	DE	SOUTH	SOUTH ATLANTIC	70195	11028	6301	30506	71.96	1
119	WASHINGTON, DC	DC	SOUTH	SOUTH ATLANTIC	638333	74287	8960	276984	36.69	1
120	CLEARWATER, FL	FL	SOUTH	SOUTH ATLANTIC	85528	22286	8223	44183	53.90	2
121	DAYTONA BEACH, FL	FL	SOUTH	SOUTH ATLANTIC	54176	11683	5879	25934	52.96	2
122	FORT LAUDERDALE, FL	FL	SOUTH	SOUTH ATLANTIC	153279	29288	9752	80107	52.96	2
123	GAINESVILLE, FL	FL	SOUTH	SOUTH ATLANTIC	81371	5699	6150	29811	43.38	2
124	HIALEAH, FL	FL	SOUTH	SOUTH ATLANTIC	145254	16508	5915	50230	52.96	1
125	HOLLYWOOD, FL	FL	SOUTH	SOUTH ATLANTIC	121323	30492	8727	58051	52.96	2

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
101	12002	29.5	73.3	15.31	3	3	2	3	2	4	1	1	1
102	6010	26.9	71.4	14.47	1	1	1	2	2	2	1	1	1
103	8582	25.0	73.6	12.68	1	1	1	1	2	3	1	1	1
104	7583	30.9	72.6	15.15	3	2	2	2	2	3	1	1	1
105	7503	29.8	76.9	10.87	3	2	1	2	1	3	1	2	1
106	10964	30.9	72.6	15.15	1	1	2	1	2	4	1	1	1
107	12043	29.5	74.0	41.56	3	3	1	2	3	4	1	1	2
108	4100	26.2	72.7	43.18	1	1	2	1	3	1	1	1	2
109	6119	26.6	71.7	48.17	1	1	2	1	3	2	1	1	3
110	17930	26.2	72.7	43.18	3	2	1	2	3	4	1	1	2
111	5128	27.4	72.5	50.10	1	2	1	1	3	1	1	1	3
112	5619	26.2	72.7	43.18	1	2	1	2	3	1	1	1	2
113	12708	29.5	74.0	41.56	3	2	1	2	3	4	1	1	2
114	2801	28.0	72.7	46.90	2	2	2	2	3	1	1	1	3
115	7601	28.0	72.7	46.90	3	2	2	2	3	3	1	1	3
116	6999	27.4	72.5	50.10	3	2	1	2	3	2	1	1	3
117	6321	29.5	74.0	41.56	1	1	1	1	3	2	1	1	2
118	11773	31.2	76.0	41.38	1	2	1	2	3	4	1	2	2
119	10692	35.2	78.9	39.00	3	3	2	3	1	4	2	2	2
120	7931	59.8	82.2	46.73	2	3	2	2	2	3	2	2	3
121	15973	57.9	81.1	48.46	1	2	1	2	2	4	2	2	3
122	13455	66.2	82.3	60.83	3	3	2	3	2	4	2	2	3
123	12067	56.4	81.4	52.84	2	1	1	2	2	4	2	2	3
124	6867	65.7	81.7	63.47	3	2	1	2	2	2	2	2	3
125	10835	66.2	82.3	60.83	3	3	2	2	2	4	2	2	3

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 15

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
126	JACKSONVILLE, FL	FL	SOUTH	SOUTH ATLANTIC	540920	52029	6767	213556	54.83	1
127	LARGO, FL	FL	SOUTH	SOUTH ATLANTIC	58977	17364	7568	31366	53.90	2
128	MIAMI, FL	FL	SOUTH	SOUTH ATLANTIC	346865	59119	6084	145762	52.96	2
129	MIAMI BEACH, FL	FL	SOUTH	SOUTH ATLANTIC	96298	49836	8904	64561	52.96	2
130	ORLANDO, FL	FL	SOUTH	SOUTH ATLANTIC	128291	16319	6735	51526	43.16	1
131	PENSACOLA, FL	FL	SOUTH	SOUTH ATLANTIC	57619	7622	6881	23312	46.22	2
132	POMPANO BEACH, FL	FL	SOUTH	SOUTH ATLANTIC	52618	15663	10204	33011	52.96	2
133	ST. PETERSBURG, FL	FL	SOUTH	SOUTH ATLANTIC	238647	61488	6966	119486	53.90	2
134	TALLAHASSEE, FL	FL	SOUTH	SOUTH ATLANTIC	81548	6071	6310	33718	48.25	2
135	TAMPA, FL	FL	SOUTH	SOUTH ATLANTIC	271523	40270	6441	114189	48.96	1
136	WEST PALM BEACH, FL	FL	SOUTH	SOUTH ATLANTIC	63305	12847	7528	30024	52.96	2
137	ALBANY, GA	GA	SOUTH	SOUTH ATLANTIC	74059	6307	5527	26181	30.08	2
138	ATLANTA, GA	GA	SOUTH	SOUTH ATLANTIC	425022	49066	6539	178826	40.53	1
139	COLUMBUS, GA	GA	SOUTH	SOUTH ATLANTIC	169441	15018	6048	63565	40.53	2
140	MACON, GA	GA	SOUTH	SOUTH ATLANTIC	116896	13963	5936	44391	40.53	1
141	SAVANNAH, GA	GA	SOUTH	SOUTH ATLANTIC	141390	16133	5896	54284	50.79	2
142	HONOLULU (CDP), HI	HI	WEST	PACIFIC	365048	38011	8948	142280	90.70	1
143	BOISE CITY, ID	ID	WEST	MOUNTAIN	102451	10484	8264	43330	21.56	1
144	ARLINGTON HEIGHTS VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	66116	5114	11136	23202	44.23	2
145	AURORA, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	81293	8059	7842	29413	44.23	1
146	CHAMPAIGN, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	58133	4342	7300	22543	50.88	2
147	CHICAGO, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	3005072	342511	6933	1174706	46.30	1
148	CICERO TOWN, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	61232	9737	7528	25870	44.23	1
149	DECATUR, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	94081	12353	7960	38456	49.40	2
150	DES PLAINES, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	53568	5615	9625	19287	44.23	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
126	7836	53.2	81.3	52.77	3	3	1	3	2	3	2	2	3
127	4049	59.8	82.2	46.73	1	3	1	2	2	1	2	2	3
128	14832	67.1	82.4	57.55	3	3	1	3	2	4	2	2	3
129	11496	68.0	82.6	46.05	2	3	2	3	2	4	2	2	3
130	12863	60.5	82.4	47.83	3	2	1	2	2	4	2	2	3
131	8128	51.7	82.3	61.16	1	2	1	1	2	3	2	2	3
132	12797	66.8	82.1	61.38	1	2	2	2	2	4	2	2	3
133	8420	61.9	83.1	53.10	3	3	1	3	2	3	2	2	3
134	11400	51.6	81.2	64.59	2	1	1	2	2	4	2	2	3
135	14309	59.8	82.2	46.73	3	3	1	3	2	4	2	2	3
136	17131	65.2	82.0	59.72	1	2	1	2	2	4	2	2	3
137	8061	48.7	81.4	49.76	1	1	1	2	1	3	2	2	3
138	13904	41.9	78.6	48.61	3	3	1	3	2	4	2	2	3
139	5048	46.2	81.0	51.09	3	2	1	3	2	1	2	2	3
140	7518	46.6	81.4	44.86	3	2	1	2	2	3	2	2	3
141	10763	49.2	81.2	49.70	3	2	1	2	2	4	2	2	3
142	6407	72.6	80.1	23.47	3	3	2	3	3	2	2	2	2
143	7317	29.9	74.6	11.71	3	2	2	2	1	2	1	1	1
144	4149	21.4	73.0	33.34	1	1	2	1	2	1	1	1	2
145	8854	20.1	73.0	35.62	2	2	2	2	2	3	1	1	2
146	9038	24.7	75.2	37.04	1	1	1	1	2	3	1	2	2
147	5753	21.4	73.0	33.34	3	3	1	3	2	1	1	1	2
148	6124	22.6	75.0	35.51	1	2	1	2	2	2	1	1	2
149	7113	25.8	76.6	39.12	2	2	2	2	2	2	1	2	2
150	4143	21.4	73.0	33.34	1	1	2	1	2	1	1	1	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 16

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
151	EAST ST. LOUIS, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	55200	4902	3681	18895	39.35	1
152	ELGIN, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	63798	7668	8146	24897	44.23	2
153	EVANSTON, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	73706	10410	11028	29295	47.95	2
154	JOLIET, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	77956	10208	7630	29816	44.23	2
155	MOUNT PROSPECT VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	52634	3745	10767	19513	44.23	2
156	OAK LAWN VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	60590	8696	9405	21191	44.23	2
157	OAK PARK VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	54887	7754	10110	23442	47.51	2
158	PEORIA, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	124160	15281	8422	50871	49.05	2
159	ROCKFORD, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	139712	17657	7849	54674	44.23	1
160	SCHAUMBURG VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	53305	2513	10053	20931	44.23	2
161	SKOKIE VILLAGE, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	60278	8788	12081	22809	44.23	2
162	SPRINGFIELD, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	99637	14294	8236	44030	33.90	3
163	WAUKEGAN, IL	IL	NORTH CENTRAL	EAST NORTH CENTRAL	67653	6720	7650	25679	47.95	1
164	ANDERSON, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	64695	8153	6843	26595	29.05	1
165	BLOOMINGTON, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	52044	3396	5804	17708	46.47	1
166	EVANSVILLE, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	130496	19888	7040	54210	35.96	1
167	FORT WAYNE, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	172196	20479	7259	70607	38.22	1
168	GARY, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	151953	12488	6171	54446	59.99	1
169	HAMMOND, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	93714	9996	7799	36103	59.99	1
170	INDIANAPOLIS, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	700807	72184	7585	283322	33.86	1
171	MUNCIE, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	77216	8348	6225	29455	38.22	1
172	SOUTH BEND, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	109727	16283	7141	44799	38.22	1

(continued on next page)

(continued from previous page)

173	TERRE HAUTE, IN	IN	NORTH CENTRAL	EAST NORTH CENTRAL	61125	9652	6068	24585	46.47	1
174	CEDAR RAPIDS, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	110243	12117	8269	43541	48.46	1
175	COUNCIL BLUFFS, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	56449	6922	6709	21949	44.98	2

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
151	5885	29.8	77.7	36.75	1	1	1	1	1	1	1	2	2
152	7087	21.4	73.0	33.34	1	2	2	1	2	2	1	1	2
153	8359	24.6	76.1	36.79	1	2	2	2	2	3	1	2	2
154	9367	21.4	73.4	35.24	2	2	2	2	2	3	1	1	2
155	4765	21.4	73.0	33.34	1	1	2	1	2	1	1	1	2
156	2723	22.6	75.0	35 51	1	2	2	1	2	1	1	1	2
157	5437	22.6	75.0	35.51	1	2	2	1	2	1	1	1	2
158	9482	21.5	75.0	34.89	3	2	2	2	2	3	1	1	2
159	9150	18.3	73.0	36.78	3	3	2	2	2	3	1	1	2
160	5443	21.4	73.0	33.34	1	1	2	1	2	1	1	1	2
161	4962	21.4	73.0	33.34	1	2	2	1	2	1	1	1	2
162	8555	24.6	76.5	33.78	2	2	2	2	1	3	1	2	2
163	9037	20.2	71.5	33.65	1	1	2	2	2	3	1	1	2
164	6571	25.6	73.8	37.54	1	2	1	2	1	2	1	1	2
165	4434	27.6	74.9	40.25	1	1	1	1	2	1	1	1	2
166	6982	30.6	78.1	41.55	3	3	1	2	1	2	1	2	2
167	7659	23.3	73.3	34.40	3	3	1	3	1	3	1	1	2
168	8072	22.9	74.0	35.48	3	2	1	2	3	3	1	1	2
169	7081	22.9	74.0	35.48	2	2	2	2	3	2	1	1	2
170	7340	26.0	75.1	39.12	3	3	2	3	1	2	1	2	2
171	7250	23.9	72.8	37.77	2	2	1	2	1	2	1	1	2
172	10840	23.2	72.5	38.16	3	2	1	2	1	4	1	1	2
173	6777	26.2	75.6	38.14	1	2	1	1	2	2	1	2	2
174	8444	18.5	74.3	36.01	3	2	2	2	2	3	1	1	2
175	10055	20.2	77.7	30.34	1	1	1	1	2	4	1	2	2

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
176	DAVENPORT, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	103264	10822	7896	40294	40.89	1
177	DES MOINES, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	191003	23879	7838	79913	45.43	2
178	DUBUQUE, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	62321	7976	7341	22160	46.41	2
179	IOWA CITY, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	50508	3313	7247	19235	40.89	2
180	SIOUX CITY, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	82003	11081	7087	32007	55.73	2
181	WATERLOO, IA	IA	NORTH CENTRAL	WEST NORTH CENTRAL	75985	8993	7872	29545	54.64	1
182	KANSAS CITY, KS	KS	NORTH CENTRAL	WEST NORTH CENTRAL	161087	18892	6399	64474	53.21	3
183	LAWRENCE, KS	KS	NORTH CENTRAL	WEST NORTH CENTRAL	52738	3399	6384	20179	46.65	2
184	OVERLAND PARK, KS	KS	NORTH CENTRAL	WEST NORTH CENTRAL	81784	5336	10623	31244	42.75	2
185	TOPEKA, KS	KS	NORTH CENTRAL	WEST NORTH CENTRAL	115266	16031	7762	50371	46.65	3
186	WICHITA, KS	KS	NORTH CENTRAL	WEST NORTH CENTRAL	279272	29704	8156	116649	39.23	2
187	LEXINGTON-FAYETTE, KY	KY	SOUTH	EAST SOUTH CENTRAL	204165	17546	7395	81747	35.79	1
188	LOUISVILLE, KY	KY	SOUTH	EAST SOUTH CENTRAL	298451	45550	6281	126143	37.40	1
189	OWENSBORO, KY	KY	SOUTH	EAST SOUTH CENTRAL	54450	6650	6550	21157	31.30	2
190	ALEXANDRIA, LA	LA	SOUTH	WEST SOUTH CENTRAL	51565	6323	6116	19637	49.01	1
191	BATON ROUGE, LA	LA	SOUTH	WEST SOUTH CENTRAL	219419	19016	7397	84080	36.38	1
192	BOSSIER CITY, LA	LA	SOUTH	WEST SOUTH CENTRAL	50817	2937	6368	17950	24.87	1
193	KENNER, LA	LA	SOUTH	WEST SOUTH CENTRAL	66382	2500	7246	22304	39.06	1
194	LAFAYETTE, LA	LA	SOUTH	WEST SOUTH CENTRAL	81961	6613	8085	29853	42.75	1
195	LAKE CHARLES, LA	LA	SOUTH	WEST SOUTH CENTRAL	75226	7430	7146	28166	36.38	1
196	MONROE, LA	LA	SOUTH	WEST SOUTH CENTRAL	57597	6979	5817	21122	39.06	1
197	NEW ORLEANS, LA	LA	SOUTH	WEST SOUTH CENTRAL	557515	65323	6463	226452	44.11	1
198	SHREVEPORT, LA	LA	SOUTH	WEST SOUTH CENTRAL	205820	24168	7211	80027	24.87	1
199	PORTLAND, ME	ME	NORTHEAST	NEW ENGLAND	61572	10200	6416	27962	50.01	2
200	BALTIMORE, MD	MD	SOUTH	SOUTH ATLANTIC	786775	100575	5877	302680	46.12	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
176	8058	20.6	75.8	33.67	3	2	2	2	2	3	1	2	2
177	10501	18.6	76.3	30.83	3	3	2	3	2	4	1	2	2
178	6813	15.6	71.9	38.58	1	2	1	1	2	2	1	1	2
179	5902	19.8	75.4	34.55	1	1	1	1	2	1	1	2	2
180	8589	16.2	75.6	25.37	2	2	1	2	3	3	1	2	2
181	7632	14.0	72.6	33.10	2	2	2	2	2	3	1	1	2
182	11204	28.4	80.9	29.27	3	3	1	3	2	4	1	2	2
183	7170	28.4	80.1	36.81	1	1	1	1	2	2	1	2	2
184	6087	28.4	80.9	29.27	2	1	2	2	2	2	1	2	2
185	9155	26.1	78.6	33.38	3	2	2	2	2	3	1	2	2
186	8986	29.6	81.4	28.61	3	3	2	3	1	3	1	2	2
187	8246	31.5	76.0	45.68	3	3	1	3	1	3	1	2	3
188	7043	32.5	77.6	43.56	3	3	1	3	1	2	2	2	2
189	5729	33.2	78.3	44.79	1	1	1	1	1	1	2	2	3
190	8885	48.4	82.5	55.90	1	1	1	1	2	3	2	2	3
191	11536	50.8	82.1	55.77	3	3	1	3	1	4	2	2	3
192	6640	46.0	82.9	43.84	1	1	1	1	1	2	2	2	2
193	6715	52.4	82.1	59.74	1	1	1	1	1	2	2	2	3
194	8695	51.7	82.1	57.69	2	1	2	2	2	3	2	2	3

(continued on next page)

(continued from previous page)

195	4854	51.5	82.3	53.03	2	2	1	2	1	1	2	2	3
196	7014	45.3	82.4	49.56	1	1	1	1	1	2	2	2	3
197	9122	52.4	82.1	59.74	3	3	1	3	2	3	2	2	3
198	8028	46.0	82.9	43.84	3	3	1	3	1	3	2	2	2
199	11409	21.5	68.1	43.52	1	2	1	2	2	4	1	1	2
200	9916	35.5	79.9	43.39	3	3	1	3	2	4	2	2	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 18

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
201	BOSTON, MA	MA	NORTHEAST	NEW ENGLAND	562994	71299	6555	241444	67.03	1
202	BROCKTON, MA	MA	NORTHEAST	NEW ENGLAND	95172	11295	6132	34720	55.19	1
203	CAMBRIDGE, MA	MA	NORTHEAST	NEW ENGLAND	95322	10871	7957	41300	46.77	2
204	CHICOPEE, MA	MA	NORTHEAST	NEW ENGLAND	55112	7532	6378	21090	42.44	1
205	FALL RIVER, MA	MA	NORTHEAST	NEW ENGLAND	92574	15375	5197	37021	55.19	1
206	LAWRENCE, MA	MA	NORTHEAST	NEW ENGLAND	63175	9593	5485	25992	50.52	3
207	LOWELL, MA	MA	NORTHEAST	NEW ENGLAND	92418	12032	6016	34883	50.52	2
208	LYNN, MA	MA	NORTHEAST	NEW ENGLAND	78471	12531	6487	32617	50.52	1
209	MALDEN, MA	MA	NORTHEAST	NEW ENGLAND	53386	8495	7165	21464	50.52	1
210	MEDFORD, MA	MA	NORTHEAST	NEW ENGLAND	58076	8588	7130	20647	50.52	2
211	NEW BEDFORD, MA	MA	NORTHEAST	NEW ENGLAND	98478	15974	5431	39523	64.28	1
212	NEWTON, MA	MA	NORTHEAST	NEW ENGLAND	83622	11881	11609	29131	67.03	1
213	PITTSFIELD, MA	MA	NORTHEAST	NEW ENGLAND	51974	7354	7105	20484	60.29	1
214	QUINCY, MA	MA	NORTHEAST	NEW ENGLAND	84743	14526	7652	34352	50.52	1
215	SOMERVILLE, MA	MA	NORTHEAST	NEW ENGLAND	77372	10495	6349	30942	67.03	1
216	SPRINGFIELD, MA	MA	NORTHEAST	NEW ENGLAND	152319	20977	5819	58692	60.29	1
217	WALTHAM, MA	MA	NORTHEAST	NEW ENGLAND	58200	7177	7666	21224	67.03	1
218	WORCESTER, MA	MA	NORTHEAST	NEW ENGLAND	161799	26325	6443	61645	50.52	2
219	ANN ARBOR, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	107966	6402	8729	40153	38.65	2
220	DEARBORN, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	90660	14157	9852	35692	38.65	1
221	DEARBORN HEIGHTS, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	67706	6424	9488	23499	38.65	1
222	DETROIT, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	1203339	140508	6215	471412	38.65	1
223	EAST LANSING, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	51392	1849	6305	13119	39.09	2
224	FARMINGTON HILLS, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	58056	4485	12632	21551	38.65	2
225	FLINT, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	159611	16019	7093	60976	37.46	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
201	14054	29.6	73.5	43.81	3	3	1	3	3	4	1	1	2
202	7800	27.0	71.3	45.77	2	2	1	2	3	3	1	1	3
203	9185	29.6	73.5	43.81	2	2	2	2	2	3	1	1	2
204	4478	26.6	73.6	44.87	1	2	1	1	2	1	1	1	3
205	6335	31.6	73.4	43.94	2	2	1	2	3	2	1	1	2
206	6529	25.8	72.6	42.56	1	2	1	2	2	2	1	1	2
207	1824	25.8	72.6	42.56	2	2	1	2	2	1	1	1	2
208	10566	29.6	73.5	43.81	2	2	1	2	2	4	1	1	2
209	3105	29.6	73.5	43.81	1	2	1	1	2	1	1	1	2
210	4039	29.6	73.5	43.81	1	2	1	1	2	1	1	1	2
211	6326	31.6	73.4	43.94	2	2	1	2	3	2	1	1	2
212	4453	29.6	73.5	43.81	2	2	2	2	3	1	1	1	2
213	5399	21.1	71.4	35.74	1	2	1	1	3	1	1	1	2
214	5422	29.6	73.5	43.81	2	2	2	2	2	1	1	1	2
215	4665	29.6	73.5	43.81	2	2	1	2	3	1	1	1	2
216	9420	26.6	73.6	44.87	3	3	1	2	3	3	1	1	3
217	1066	29.6	73.5	43.81	1	2	2	1	3	1	1	1	2
218	4563	23.3	69.9	47.60	3	3	1	3	2	1	1	1	3
219	8427	23.8	72.9	30.22	3	1	2	2	1	3	1	1	2
220	9010	23.4	71.9	30.97	2	2	2	2	1	3	1	1	2
221	6938	23.4	71.9	30.97	1	1	2	1	1	2	1	1	2
222	11987	23.4	71.9	30.97	3	3	1	3	1	4	1	1	2
223	3071	21.6	70.8	29.58	1	1	1	1	1	1	1	1	2
224	5240	14.0	64.5	30.97	1	1	2	1	1	1	1	1	2
225	14864	21.3	70.1	29.21	3	2	1	3	1	4	1	1	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 19

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
226	GRAND RAPIDS, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	181843	24435	6691	69888	37.46	2
227	KALAMAZOO, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	79722	8425	6945	30207	37.46	2
228	LANSING, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	130414	11410	7280	51948	39.09	1
229	LIVONIA, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	104814	8127	9969	33012	38.65	1
230	PONTIAC, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	76715	6945	6252	27745	38.65	2
231	ROSEVILLE, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	54311	4297	7539	18491	38.65	2
232	ROYAL OAK, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	70893	8762	9766	28785	38.65	2
233	SAGINAW, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	77508	8652	6101	28747	37.46	2
234	ST. CLAIR SHORES, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	76210	8595	9191	27154	38.65	2
235	SOUTHFIELD, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	75568	11092	12668	31289	38.65	2

(continued on next page)

(continued from previous page)

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
236	STERLING HEIGHTS, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	108999	4955	8996	34517	38.65	2
237	TAYLOR, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	77568	3975	7413	25355	38.65	1
238	TROY, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	67102	3755	11642	23750	38.65	2
239	WARREN, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	161134	12836	8690	54532	38.65	1
240	WESTLAND, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	84603	6058	8308	29963	38.65	1
241	WYOMING, MI	MI	NORTH CENTRAL	EAST NORTH CENTRAL	59616	5041	7452	22703	37.46	2
242	BLOOMINGTON, MN	MN	NORTH CENTRAL	WEST NORTH CENTRAL	81831	4589	10220	29569	43.20	2
243	DULUTH, MN	MN	NORTH CENTRAL	WEST NORTH CENTRAL	92811	14267	7176	37090	42.33	1
244	MINNEAPOLIS, MN	MN	NORTH CENTRAL	WEST NORTH CENTRAL	370951	57030	7940	168859	44.50	1
245	ROCHESTER, MN	MN	NORTH CENTRAL	WEST NORTH CENTRAL	57890	6177	8472	23110	34.07	1
246	ST. PAUL, MN	MN	NORTH CENTRAL	WEST NORTH CENTRAL	270230	40587	7694	110902	45.06	1
247	JACKSON, MS	MS	SOUTH	EAST SOUTH CENTRAL	202895	19716	6920	75644	44.30	3
248	COLUMBIA, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	62061	4720	6706	22690	48.29	2
249	FLORISSANT, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	55372	4141	7950	18048	35.95	1
250	INDEPENDENCE, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	111806	12202	7839	44397	37.22	2

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
226	9416	22.0	71.4	34.36	3	3	1	3	1	3	1	1	2
227	11077	23.7	73.2	34.83	2	2	1	2	1	4	1	1	2
228	7980	21.6	70.8	29.58	3	2	1	2	1	3	1	1	2
229	5407	20.2	67.4	30.97	3	2	2	2	1	1	1	1	2
230	11331	22.4	72.1	29.30	2	1	1	2	1	4	1	1	2
231	7219	20.0	67.9	30.97	1	1	1	1	1	2	1	1	2
232	5755	20.0	67.9	30.97	1	2	2	2	1	1	1	1	2
233	12928	20.7	71.0	29.77	2	2	1	2	1	4	1	1	2
234	4756	20.0	67.9	30.97	2	2	2	2	1	1	1	1	2
235	10428	20.0	67.9	30.97	2	2	2	2	1	4	1	1	2
236	5017	22.4	72.1	29.30	3	1	2	2	1	1	1	1	2
237	7852	23.4	72.6	30.97	2	1	1	2	1	3	1	1	2
238	5468	22.4	72.1	29.30	1	1	2	1	1	1	1	1	2
239	7526	23.4	72.6	30.97	3	2	2	2	1	3	1	1	2
240	6018	23.4	72.6	30.97	2	1	2	2	1	2	1	1	2
241	5697	22.0	71.4	34.36	1	1	1	1	1	1	1	1	2
242	5358	11.2	73.1	26.36	2	1	2	2	2	1	1	1	2
243	6006	6.3	65.3	29.68	2	2	1	2	2	2	1	1	2
244	10251	11.2	73.1	26.36	3	3	2	3	2	4	1	1	2
245	5189	10.8	70.7	28.26	1	1	2	1	1	1	1	1	2
246	8433	11.2	73.1	26.36	3	3	2	3	2	3	1	1	2
247	9192	45.7	81.9	52.82	3	3	1	3	2	3	2	2	3
248	8926	27.5	77.8	36.06	1	1	1	1	2	3	1	2	2
249	2651	28.8	78.9	33.91	1	1	2	1	1	1	1	2	2
250	5775	28.4	80.9	29.27	3	2	2	2	1	1	1	2	2

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
251	KANSAS CITY, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	448159	55148	7480	191904	44.54	2
252	ST. JOSEPH, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	76691	12201	6367	31945	45.83	1
253	ST. LOUIS, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	453085	79920	5879	202113	35.95	1
254	SPRINGFIELD, MO	MO	NORTH CENTRAL	WEST NORTH CENTRAL	133116	17589	6468	56078	41.09	2
255	BILLINGS, MT	MT	WEST	MOUNTAIN	66798	7237	7947	28000	24.43	2
256	GREAT FALLS, MT	MT	WEST	MOUNTAIN	56725	6514	7369	24000	24.43	2
257	LINCOLN, NE	NE	NORTH CENTRAL	WEST NORTH CENTRAL	171932	17746	7631	69138	35.24	1
258	OMAHA, NE	NE	NORTH CENTRAL	WEST NORTH CENTRAL	314255	38426	7529	125445	31.15	1
259	LAS VEGAS, NV	NV	WEST	MOUNTAIN	164674	13750	8135	67133	36.71	2
260	RENO, NV	NV	WEST	MOUNTAIN	100756	10577	9620	47380	55.50	2
261	MANCHESTER, NH	NH	NORTHEAST	NEW ENGLAND	90936	12202	6841	35869	57.57	1
262	NASHUA, NH	NH	NORTHEAST	NEW ENGLAND	67865	6429	7844	25444	57.57	1
263	BAYONNE, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	65047	9672	7751	26363	63.81	1
264	CAMDEN, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	84910	8602	3966	32573	63.81	1
265	CLIFTON, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	74388	12557	8803	29440	63.81	2
266	EAST ORANGE, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	77690	8022	6238	31081	63.81	1
267	ELIZABETH, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	106201	13973	6712	40613	63.81	1
268	IRVINGTON TOWN, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	61493	8897	6611	25527	63.81	1
269	JERSEY CITY, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	223532	26283	5812	87999	63.81	1
270	NEWARK, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	329248	28838	4525	121387	63.81	1
271	PASSAIC, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	52463	6500	5813	19857	63.81	2
272	PATERSON, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	137970	14274	5060	48159	63.81	1
273	TRENTON, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	92124	11621	5400	35819	63.81	1
274	UNION CITY, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	55593	6400	5625	21500	63.81	3
275	VINELAND, NJ	NJ	NORTHEAST	MIDDLE ATLANTIC	53753	6217	6153	18120	54.02	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
251	11329	28.4	80.9	29.27	3	3	1	3	2	4	1	2	2
252	7353	24.9	78.5	34.51	2	2	1	2	2	2	1	2	2
253	13795	28.8	78.9	33.91	3	3	1	3	1	4	1	2	2
254	10009	31.5	78.0	39.47	3	3	1	2	2	4	1	2	2
255	6420	20.9	72.3	15.09	1	2	2	2	1	2	1	1	1
256	8044	18.7	69.3	15.24	1	1	1	1	1	3	1	1	1

(continued on next page)

(continued from previous page)

257	6122	21.9	78.5	28.36	3	3	2	3	1	2	1	2	2
258	7542	20.2	77.7	30.34	3	3	1	3	1	3	1	2	2
259	10660	44.6	90.3	4.19	3	2	2	3	1	4	2	2	1
260	9172	32.2	69.5	7.49	3	2	2	2	3	3	2	1	1
261	7030	22.6	69.8	43.27	2	2	1	2	3	2	1	1	2
262	5239	22.6	69.8	43.27	1	1	2	2	3	1	1	1	2
263	3559	30.6	74.6	43.77	1	2	2	2	3	1	1	1	2
264	15715	31.2	76.5	41.42	2	2	1	2	3	4	1	2	2
265	4839	31.3	76.8	42.34	1	2	2	2	3	1	1	2	2
266	11355	31.3	76.8	42.34	2	2	1	2	3	4	1	2	2
267	8764	30.6	74.6	43.77	3	2	1	2	3	3	1	1	2
268	9157	31.3	76.8	42.34	1	2	1	2	3	3	1	2	2
269	8873	30.6	74.6	43.77	3	3	1	3	3	3	1	1	2
270	12318	31.3	76.8	42.34	3	3	1	3	3	4	1	2	2
271	9488	31.3	76.8	42.34	1	1	1	1	3	3	1	2	2
272	10575	31.3	76.8	42.34	3	2	1	2	3	4	1	2	2
273	10711	31.6	75.9	42.40	2	2	1	2	3	4	1	2	2
274	7478	30.6	74.6	43.77	1	1	1	1	3	2	1	1	2
275	10031	31.8	75.8	42.95	1	1	1	1	2	4	1	2	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 21

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
276	ALBUQUERQUE, NM	NM	WEST	MOUNTAIN	331767	27966	7439	132788	63.03	1
277	ALBANY, NY	NY	NORTHEAST	MIDDLE ATLANTIC	101727	16766	6708	46209	40.95	1
278	BINGHAMTON, NY	NY	NORTHEAST	MIDDLE ATLANTIC	55860	10206	6495	24247	51.13	1
279	BUFFALO, NY	NY	NORTHEAST	MIDDLE ATLANTIC	357870	53724	5929	156470	40.95	1
280	MOUNT VERNON, NY	NY	NORTHEAST	MIDDLE ATLANTIC	66713	9767	7492	26189	89.38	1
281	NEW ROCHELLE, NY	NY	NORTHEAST	MIDDLE ATLANTIC	70794	10901	10343	26225	89.38	2
282	NEW YORK, NY	NY	NORTHEAST	MIDDLE ATLANTIC	7071639	951732	7271	2946410	90.53	1
283	NIAGARA FALLS, NY	NY	NORTHEAST	MIDDLE ATLANTIC	71384	10541	6443	29504	40.95	2
284	ROCHESTER, NY	NY	NORTHEAST	MIDDLE ATLANTIC	241741	33811	6492	102642	51.23	2
285	SCHENECTADY, NY	NY	NORTHEAST	MIDDLE ATLANTIC	67972	11385	6494	30249	40.95	1
286	SYRACUSE, NY	NY	NORTHEAST	MIDDLE ATLANTIC	170105	24833	6232	73175	40.95	1
287	TROY, NY	NY	NORTHEAST	MIDDLE ATLANTIC	56638	8196	5536	22587	40.95	2
288	UTICA, NY	NY	NORTHEAST	MIDDLE ATLANTIC	75632	13486	5592	31796	40.95	1
289	YONKERS, NY	NY	NORTHEAST	MIDDLE ATLANTIC	195351	28943	8339	75907	89.38	2
290	ASHEVILLE, NC	NC	SOUTH	SOUTH ATLANTIC	53583	9797	6533	23239	48.74	2
291	CHARLOTTE, NC	NC	SOUTH	SOUTH ATLANTIC	314447	27167	7820	124069	41.34	2
292	DURHAM, NC	NC	SOUTH	SOUTH ATLANTIC	100831	12188	6366	39768	41.34	2
293	FAYETTEVILLE, NC	NC	SOUTH	SOUTH ATLANTIC	59507	5157	6166	23053	43.38	2
294	GREENSBORO, NC	NC	SOUTH	SOUTH ATLANTIC	155642	15287	7502	59859	41.34	2
295	HIGH POINT, NC	NC	SOUTH	SOUTH ATLANTIC	63380	8263	6678	24320	39.95	2
296	RALEIGH, NC	NC	SOUTH	SOUTH ATLANTIC	150255	12438	7804	57866	48.74	2
297	WINSTON-SALEM, NC	NC	SOUTH	SOUTH ATLANTIC	131885	15967	7329	53597	41.34	2
298	FARGO, ND	ND	NORTH CENTRAL	WEST NORTH CENTRAL	61383	6037	7950	25219	36.72	3
299	AKRON, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	237177	31970	6784	96682	51.79	1
300	CANTON, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	94730	13655	6373	39254	38.32	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
276	9089	34.8	78.8	8.12	3	3	1	3	3	3	2	2	1
277	5789	21.1	71.4	35.74	3	2	1	2	2	1	1	1	2
278	5018	21.2	68.9	36.79	1	2	1	1	2	1	1	1	2
279	7435	23.5	70.7	37.52	3	3	1	3	2	2	1	1	2
280	7935	31.8	76.7	44.12	1	2	1	2	3	3	1	2	3
281	4857	29.6	74.1	47.42	1	2	2	2	3	1	1	1	3
282	10280	31.8	76.7	44.12	3	3	1	3	3	4	1	2	3
283	6926	23.5	70.7	37.52	1	2	1	2	2	2	1	1	2
284	11243	23.6	71.3	31.27	3	3	1	3	2	4	1	1	2
285	4779	21.1	71.4	35.74	1	2	1	2	2	1	1	1	2
286	8440	22.8	70.9	39.11	3	3	1	3	2	3	1	1	2
287	5747	21.1	71.4	35.74	1	2	1	1	2	1	1	1	2
288	3659	20.1	69.8	43.44	2	2	1	2	2	1	1	1	2
289	5737	31.8	76.7	44.12	3	3	2	3	3	1	1	2	3
290	6781	36.8	73.2	47.72	1	2	1	1	2	2	2	1	3
291	9367	40.5	78.5	43.16	3	3	2	3	2	3	2	2	2
292	10581	39.5	77.7	47.06	3	2	1	2	2	4	2	2	3
293	11641	41.4	79.3	47.67	1	1	1	1	2	4	2	2	3
294	7617	37.5	77.2	42.47	3	2	1	2	2	3	2	2	2
295	7737	39.8	77.6	44.15	1	2	1	1	1	3	2	2	3
296	7334	39.6	77.7	41.76	3	2	2	2	2	2	2	2	2
297	9353	39.8	77.6	44.15	3	2	1	2	2	3	2	2	3
298	5522	4.3	70.6	19.59	1	1	2	2	1	1	1	1	2
299	7810	25.1	71.6	35.90	3	3	1	3	2	3	1	1	2
300	6612	25.1	71.6	35.90	2	2	1	2	1	2	1	1	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 22

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
301	CINCINNATI, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	385457	55712	6875	172659	39.04	2
302	CLEVELAND, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	573822	74431	5770	239557	62.39	1
303	CLEVELAND HEIGHTS, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	56438	7483	9233	21405	57.02	2
304	COLUMBUS, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	564871	50163	6783	236708	51.55	1
305	DAYTON, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	203371	24052	5771	86789	55.39	2
306	ELYRIA, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	57538	5313	7208	22010	51.79	1
307	EUCLID, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	59999	10073	8693	26417	57.02	1
308	HAMILTON, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	63189	8078	6634	24951	37.82	2
309	KETTERING, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	61186	7387	9794	25339	55.39	2
310	LAKEWOOD, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	61963	9363	9008	28458	57.02	1
311	LORAIN, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	75416	7565	6727	27594	51.79	1
312	MANSFIELD, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	53927	7173	6536	22469	51.79	1
313	PARMA, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	92548	11640	8503	34287	57.02	1
314	SPRINGFIELD, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	72563	10220	6283	29347	51.79	2
315	TOLEDO, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	354635	44339	7050	143296	55.68	2
316	WARREN, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	56629	7430	7166	22363	51.79	1
317	YOUNGSTOWN, OH	OH	NORTH CENTRAL	EAST NORTH CENTRAL	115436	16802	5913	45105	51.79	1
318	ENID, OK	OK	SOUTH	WEST SOUTH CENTRAL	50363	6669	7822	20790	39.06	2
319	LAWTON, OK	OK	SOUTH	WEST SOUTH CENTRAL	80054	5643	5904	31911	31.68	2
320	NORMAN, OK	OK	SOUTH	WEST SOUTH CENTRAL	68020	4925	7410	26875	38.67	2
321	OKLAHOMA CITY, OK	OK	SOUTH	WEST SOUTH CENTRAL	403213	45368	7991	177136	39.06	2
322	TULSA, OK	OK	SOUTH	WEST SOUTH CENTRAL	360919	38825	8841	156353	31.68	3
323	EUGENE, OR	OR	WEST	PACIFIC	105624	10005	7819	44942	21.28	2
324	PORTLAND, OR	OR	WEST	PACIFIC	366383	55929	8092	167911	24.38	3
325	SALEM, OR	OR	WEST	PACIFIC	89233	11816	7310	37125	24.38	2

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
301	9601	30.3	76.1	40.10	3	3	1	3	1	4	1	2	2
302	10594	25.5	71.6	35.40	3	3	1	3	3	4	1	1	2
303	4958	25.5	71.6	35.40	1	2	2	1	3	1	1	1	2
304	9823	27.1	73.8	36.97	3	3	1	3	2	4	1	1	2
305	13982	26.6	74.7	34.71	3	3	1	3	3	4	1	1	2
306	4404	25.9	72.6	35.08	1	1	1	1	2	1	1	1	2
307	3778	25.5	71.6	35.40	1	2	2	2	3	1	1	1	2
308	11660	29.9	75.4	38.20	1	2	1	1	1	4	1	2	2
309	5061	26.6	74.7	34.71	1	2	2	2	3	1	1	1	2
310	2306	25.5	71.6	35.40	1	2	2	2	3	1	1	1	2
311	2650	25.9	72.6	35.08	2	2	1	2	2	1	1	1	2
312	9875	24.8	72.0	34.88	1	2	1	1	2	4	1	1	2
313	3300	25.5	71.6	35.40	2	2	2	2	3	1	1	1	2
314	7103	26.6	74.7	34.71	1	2	1	2	2	2	1	1	2
315	9607	23.1	71.8	31.77	3	3	1	3	3	4	1	1	2
316	6573	25.6	70.9	35.03	1	2	1	1	2	2	1	1	2
317	8418	24.2	70.1	37.33	3	2	1	2	2	3	1	1	2
318	6340	35.4	83.5	31.23	1	1	2	1	1	2	2	2	2
319	6695	38.8	83.7	29.20	2	1	1	2	1	2	2	2	2
320	5273	37.4	81.7	30.89	1	1	1	2	1	1	2	2	2
321	8474	35.9	82.1	30.89	3	3	2	3	1	3	2	2	2
322	8239	35.2	83.2	38.77	3	3	2	3	1	3	2	2	2
323	10029	40.1	66.8	46.04	3	2	2	2	1	4	2	1	3
324	13648	38.9	67.7	37.39	3	3	2	3	1	4	2	1	2
325	9329	39.3	66.3	40.35	2	2	1	2	1	3	2	1	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 23

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
326	ALLENTOWN, PA	PA	NORTHEAST	MIDDLE ATLANTIC	103758	16696	7134	43763	47.66	1
327	ALTOONA, PA	PA	NORTHEAST	MIDDLE ATLANTIC	57078	9437	5874	22502	50.23	1
328	BETHLEHEM, PA	PA	NORTHEAST	MIDDLE ATLANTIC	70419	9969	7556	27395	47.66	1
329	ERIE, PA	PA	NORTHEAST	MIDDLE ATLANTIC	119123	15940	6218	46851	50.23	1
330	HARRISBURG, PA	PA	NORTHEAST	MIDDLE ATLANTIC	53264	8128	6190	26034	47.66	1
331	LANCASTER, PA	PA	NORTHEAST	MIDDLE ATLANTIC	54725	7418	5594	21959	47.66	1
332	PHILADELPHIA, PA	PA	NORTHEAST	MIDDLE ATLANTIC	1688210	237370	6053	685629	57.05	1
333	PITTSBURGH, PA	PA	NORTHEAST	MIDDLE ATLANTIC	423938	67929	6845	179191	55.74	1
334	READING, PA	PA	NORTHEAST	MIDDLE ATLANTIC	78686	14020	6083	34142	53.87	3
335	SCRANTON, PA	PA	NORTHEAST	MIDDLE ATLANTIC	88117	16601	5780	36159	47.66	1
336	WILKES-BARRE, PA	PA	NORTHEAST	MIDDLE ATLANTIC	51551	9673	5610	21395	47.66	1
337	CRANSTON, RI	RI	NORTHEAST	NEW ENGLAND	71992	11704	7512	27280	51.91	1
338	EAST PROVIDENCE, RI	RI	NORTHEAST	NEW ENGLAND	50980	8015	6879	19402	51.91	2
339	PAWTUCKET, RI	RI	NORTHEAST	NEW ENGLAND	71204	11490	6328	29768	66.25	1
340	PROVIDENCE, RI	RI	NORTHEAST	NEW ENGLAND	156804	24057	6169	67535	51.91	1
341	WARWICK, RI	RI	NORTHEAST	NEW ENGLAND	87123	11267	7540	32450	51.91	1
342	CHARLESTON, SC	SC	SOUTH	SOUTH ATLANTIC	69510	8497	6906	27255	49.55	1
343	COLUMBIA, SC	SC	SOUTH	SOUTH ATLANTIC	101208	10489	5864	32564	49.55	2
344	GREENVILLE, SC	SC	SOUTH	SOUTH ATLANTIC	58242	8010	6924	23487	42.32	2
345	NORTH CHARLESTON, SC	SC	SOUTH	SOUTH ATLANTIC	62534	2698	5737	20222	49.55	1
346	SIOUX FALLS, SD	SD	NORTH CENTRAL	WEST NORTH CENTRAL	81343	9015	7527	32984	40.52	3
347	CHATTANOOGA, TN	TN	SOUTH	EAST SOUTH CENTRAL	169565	21565	6328	66630	34.28	3
348	CLARKSVILLE, TN	TN	SOUTH	EAST SOUTH CENTRAL	54777	3585	5716	19412	34.27	1

(continued on next page)

(continued from previous page)

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
349	KNOXVILLE, TN	TN	SOUTH	EAST SOUTH CENTRAL	175030	24172	6378	73263	34.27	1
350	MEMPHIS, TN	TN	SOUTH	EAST SOUTH CENTRAL	646356	67419	6466	244470	34.27	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
326	6510	27.2	73.8	44.31	3	2	1	2	2	2	1	1	3
327	2402	27.9	72.9	45.66	1	2	1	1	2	1	1	1	3
328	4319	27.2	73.8	44.31	1	2	1	2	2	1	1	1	3
329	4761	24.5	69.1	39.39	3	2	1	2	2	1	1	1	2
330	11893	29.4	75.8	39.09	1	2	1	2	2	4	1	2	2
331	7469	30.0	74.7	40.63	1	2	1	1	2	2	1	1	2
332	5966	31.2	76.5	41.42	3	3	1	3	3	1	1	2	2
333	7374	26.7	72.0	36.29	3	3	1	3	3	2	1	1	2
334	6488	25.8	70.8	48.76	2	2	1	2	2	2	1	1	3
335	4497	25.2	71.8	35.08	2	2	1	2	2	1	1	1	2
336	3496	25.2	71.8	35.08	1	2	1	1	2	1	1	1	2
337	5363	28.2	72.5	45.32	1	2	1	2	2	1	1	1	3
338	4223	28.2	72.5	45.32	1	2	1	1	2	1	1	1	3
339	5251	28.2	72.5	45.32	1	2	1	2	3	1	1	1	3
340	9869	28.2	72.5	45.32	3	3	1	3	2	4	1	1	3
341	6749	28.2	72.5	45.32	2	2	1	2	2	2	1	1	3
342	9480	49.2	81.6	47.34	1	2	1	2	2	3	2	2	3
343	12768	44.7	81.0	49.12	3	2	1	2	2	4	2	2	3
344	10358	41.1	78.2	50.52	1	2	1	1	2	4	2	2	3
345	9525	49.2	81.6	47.34	1	1	1	1	2	4	2	2	3
346	5369	12.4	74.0	24.12	2	2	1	2	2	1	1	1	2
347	8072	38.7	78.7	52.60	3	3	1	3	1	3	2	2	3
348	4596	35.7	78.7	49.64	1	1	1	1	1	1	2	2	3
349	6778	38.2	77.6	47.29	3	3	1	3	1	2	2	2	3
350	8152	39.6	82.1	51.57	3	3	1	3	1	3	2	2	3

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 24

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
351	NASHVILLE-DAVIDSON, TN	TN	SOUTH	EAST SOUTH CENTRAL	455651	50054	7276	179129	32.48	1
352	ABILENE, TX	TX	SOUTH	WEST SOUTH CENTRAL	98315	10086	6808	36497	41.66	2
353	AMARILLO, TX	TX	SOUTH	WEST SOUTH CENTRAL	149230	15128	7776	60280	46.56	2
354	ARLINGTON, TX	TX	SOUTH	WEST SOUTH CENTRAL	160113	7134	8823	65352	47.87	2
355	AUSTIN, TX	TX	SOUTH	WEST SOUTH CENTRAL	345496	25862	7247	146503	29.70	2
356	BAYTOWN, TX	TX	SOUTH	WEST SOUTH CENTRAL	56923	4652	8384	21841	48.63	2
357	BEAUMONT, TX	TX	SOUTH	WEST SOUTH CENTRAL	118102	13509	7581	47065	46.60	2
358	BROWNSVILLE, TX	TX	SOUTH	WEST SOUTH CENTRAL	84997	7126	4129	24430	46.38	2
359	CORPUS CHRISTI, TX	TX	SOUTH	WEST SOUTH CENTRAL	231999	19107	6821	81587	45.64	2
360	DALLAS, TX	TX	SOUTH	WEST SOUTH CENTRAL	904078	85975	8612	390426	44.39	2
361	EL PASO, TX	TX	SOUTH	WEST SOUTH CENTRAL	425259	29532	5439	134368	51.99	1
362	FORT WORTH, TX	TX	SOUTH	WEST SOUTH CENTRAL	385164	45435	7336	156031	47.87	2
363	GALVESTON, TX	TX	SOUTH	WEST SOUTH CENTRAL	61902	8106	7292	27850	48.63	2
364	GARLAND, TX	TX	SOUTH	WEST SOUTH CENTRAL	138857	5644	8108	48328	46.32	2
365	GRAND PRAIRIE, TX	TX	SOUTH	WEST SOUTH CENTRAL	71462	4064	7232	25091	47.87	2
366	HOUSTON, TX	TX	SOUTH	WEST SOUTH CENTRAL	1595138	109726	8793	678324	48.63	1
367	IRVING, TX	TX	SOUTH	WEST SOUTH CENTRAL	109943	5183	8498	43100	34.09	2
368	LAREDO, TX	TX	SOUTH	WEST SOUTH CENTRAL	91449	7884	3714	25250	45.64	1
369	LONGVIEW, TX	TX	SOUTH	WEST SOUTH CENTRAL	62762	6677	7412	25106	30.60	2
370	LUBBOCK, TX	TX	SOUTH	WEST SOUTH CENTRAL	173979	13491	7112	66951	55.21	2
371	MCALLEN, TX	TX	SOUTH	WEST SOUTH CENTRAL	66281	6336	5668	21834	45.64	2
372	MESQUITE, TX	TX	SOUTH	WEST SOUTH CENTRAL	67053	3009	7330	22253	34.09	2
373	MIDLAND, TX	TX	SOUTH	WEST SOUTH CENTRAL	70525	5517	10386	26947	47.87	2
374	ODESSA, TX	TX	SOUTH	WEST SOUTH CENTRAL	90027	6634	7970	33450	47.87	2
375	PASADENA, TX	TX	SOUTH	WEST SOUTH CENTRAL	112560	5385	8192	44355	48.63	1

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
351	7330	37.1	79.4	48.49	3	3	1	3	1	2	2	2	3
352	5560	43.3	84.1	23.26	2	2	1	2	2	1	2	2	2
353	6278	35.4	78.8	19.10	3	2	2	3	2	2	2	2	2
354	6876	44.0	86.3	29.45	3	2	2	3	2	2	2	2	2
355	8632	49.1	84.7	31.50	3	3	1	3	1	3	2	2	2
356	7821	51.4	83.1	44.77	1	1	2	1	2	3	2	2	3
357	10774	51.9	83.1	52.79	3	2	2	2	2	4	2	2	3
358	8594	60.3	84.1	25.44	2	2	1	1	2	3	2	2	2
359	8831	56.3	84.9	30.18	3	3	1	3	2	3	2	2	2
360	11905	45.0	86.3	34.16	3	3	2	3	2	4	2	2	2
361	6613	44.2	82.5	7.82	3	3	1	3	2	2	2	2	1
362	11833	44.0	86.3	29.45	3	3	1	3	2	4	2	2	2
363	9672	53.6	83.2	40.24	1	2	1	2	2	4	2	2	2
364	4878	45.0	86.3	34.16	3	1	2	2	2	1	2	2	2
365	7721	44.0	86.3	29.45	1	1	1	2	2	3	2	2	2
366	6143	51.4	83.1	44.77	3	3	2	3	2	2	2	2	3
367	7310	45.0	86.3	34.16	3	1	2	2	1	2	2	2	2
368	6442	56.2	87.4	20.14	2	2	1	2	2	2	2	2	2
369	6551	44.1	82.8	46.41	1	1	1	2	1	2	2	2	3
370	8501	38.8	79.8	17.76	3	2	1	3	3	3	2	2	1

(continued on next page)

(continued from previous page)

371	8052	58.8	84.4	23.04	1	1	1	1	2	3	2	2	2
372	6899	45.0	86.3	34.16	1	1	1	1	1	2	2	2	2
373	4100	43.7	81.7	13.70	1	1	2	2	2	1	2	2	1
374	9143	43.7	81.7	13.70	2	1	2	2	2	3	2	2	1
375	6917	51.4	83.1	44.77	3	1	2	2	2	2	2	2	3

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 25

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT
376	PLANO, TX	TX	SOUTH	WEST SOUTH CENTRAL	72331	1945	9178	24261	34.09	2
377	PORT ARTHUR, TX	TX	SOUTH	WEST SOUTH CENTRAL	61251	8286	6538	24133	46.60	2
378	RICHARDSON, TX	TX	SOUTH	WEST SOUTH CENTRAL	72496	2826	10690	25391	34.09	2
379	SAN ANGELO, TX	TX	SOUTH	WEST SOUTH CENTRAL	73240	9053	6710	28331	41.66	2
380	SAN ANTONIO, TX	TX	SOUTH	WEST SOUTH CENTRAL	785880	74506	5671	277803	42.78	2
381	TYLER, TX	TX	SOUTH	WEST SOUTH CENTRAL	70508	9219	7451	28404	34.09	2
382	VICTORIA, TX	TX	SOUTH	WEST SOUTH CENTRAL	50695	4406	7108	18235	45.64	2
383	WACO, TX	TX	SOUTH	WEST SOUTH CENTRAL	101261	14431	6106	39997	34.09	2
384	WICHITA FALLS, TX	TX	SOUTH	WEST SOUTH CENTRAL	94201	10556	7108	37949	47.87	2
385	OGDEN, UT	UT	WEST	MOUNTAIN	64407	8374	6539	25675	50.07	2
386	OREM, UT	UT	WEST	MOUNTAIN	52399	2312	5311	14826	49.60	2
387	PROVO, UT	UT	WEST	MOUNTAIN	74108	4403	4814	21284	32.47	1
388	SALT LAKE CITY, UT	UT	WEST	MOUNTAIN	163033	24025	7409	72830	50.07	3
389	SANDY CITY, UT	UT	WEST	MOUNTAIN	50546	1062	6505	13682	49.60	1
390	ALEXANDRIA, VA	VA	SOUTH	SOUTH ATLANTIC	103217	9465	12177	52041	51.81	2
391	CHESAPEAKE, VA	VA	SOUTH	SOUTH ATLANTIC	114486	8088	6646	38060	51.81	2
392	HAMPTON, VA	VA	SOUTH	SOUTH ATLANTIC	122617	8525	6786	43671	51.81	2
393	LYNCHBURG, VA	VA	SOUTH	SOUTH ATLANTIC	66743	9324	6896	25421	41.27	2
394	NEWPORT NEWS, VA	VA	SOUTH	SOUTH ATLANTIC	144903	11372	6856	54994	51.81	2
395	NORFOLK, VA	VA	SOUTH	SOUTH ATLANTIC	266979	24475	6113	94871	51.81	2
396	PORTSMOUTH, VA	VA	SOUTH	SOUTH ATLANTIC	104577	11242	6104	38611	51.81	2
397	RICHMOND, VA	VA	SOUTH	SOUTH ATLANTIC	219214	30838	7073	91527	51.81	2
398	ROANOKE, VA	VA	SOUTH	SOUTH ATLANTIC	100220	15678	6816	42690	41.27	2
399	VIRGINIA BEACH, VA	VA	SOUTH	SOUTH ATLANTIC	262199	11913	7704	92032	51.81	2
400	BELLEVUE, WA	WA	WEST	PACIFIC	73903	4556	11666	29315	20.67	2

OBS	CRIME	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
376	4425	45.0	86.3	34.16	1	1	2	1	1	1	2	2	2
377	5836	51.9	83.1	52.79	1	2	1	1	2	1	2	2	3
378	5468	45.0	86.3	34.16	1	1	2	2	1	1	2	2	2
379	6385	45.5	84.3	18.19	1	2	1	2	2	2	2	2	1
380	7560	50.4	84.6	29.13	3	3	1	3	2	3	2	2	2
381	8751	45.8	82.4	44.74	1	2	1	2	1	3	2	2	3
382	6707	53.4	84.5	36.90	1	1	1	1	2	2	2	2	2
383	8251	46.2	85.9	30.95	3	2	1	2	1	3	2	2	2
384	7517	40.3	85.6	26.73	2	2	1	2	2	3	2	2	2
385	10135	28.6	77.0	20.58	1	2	1	2	2	4	1	2	2
386	3946	28.6	77.5	15.31	1	1	1	1	2	1	1	2	1
387	3477	28.6	77.5	15.31	1	1	1	1	1	1	1	2	1
388	12309	28.6	77.5	15.31	3	3	1	3	2	4	1	2	1
389	5206	28.6	77.5	15.31	1	1	1	1	2	1	1	2	1
390	8764	35.2	78.9	39.00	3	2	2	2	2	3	2	2	2
391	3906	38.9	77.9	46.93	3	2	1	2	2	1	2	2	3
392	6045	39.9	78.4	45.22	3	2	1	2	2	2	2	2	3
393	6445	35.1	75.7	39.91	1	2	1	2	2	2	2	2	2
394	6291	40.4	79.7	45.21	3	2	1	2	2	2	2	2	3
395	7777	39.9	78.4	45.22	3	3	1	3	2	3	2	2	3
396	5949	38.9	77.9	46.93	3	2	1	2	2	1	2	2	3
397	11625	36.6	77.8	44.07	3	3	1	3	2	4	2	2	3
398	9703	35.5	75.7	39.15	3	2	1	2	2	4	2	2	2
399	5785	39.9	78.4	45.22	3	2	2	3	2	1	2	2	3
400	7299	40.6	65.3	38.85	1	1	2	2	1	2	2	1	2

CENSUS DATA: OUTPUT FROM PRINT PROCEDURE 26

OBS	CITY	ST	REGION	DIVISION	POP1980	OVER65	INCOME	HOUSING	ELECTRIC	CITYGOVT	CRIME
401	EVERETT, WA	WA	WEST	PACIFIC	54413	7861	7819	23912	24.65	1	10910
402	SEATTLE, WA	WA	WEST	PACIFIC	493846	76174	9282	230039	10.34	1	11071
403	SPOKANE, WA	WA	WEST	PACIFIC	171300	26166	7161	76041	18.29	2	8965
404	TACOMA, WA	WA	WEST	PACIFIC	158501	21460	7050	67759	16.15	2	10002
405	CHARLESTON, WV	WV	SOUTH	SOUTH ATLANTIC	63968	10375	8946	28027	35.99	1	13065
406	HUNTINGTON, WV	WV	SOUTH	SOUTH ATLANTIC	63684	10865	6867	27631	35.99	2	8172
407	APPLETON, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	59032	6560	7861	21626	38.50	1	6283
408	EAU CLAIRE, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	51509	6047	6345	19224	36.47	2	5172
409	GREEN BAY, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	87899	10511	6991	34445	37.85	1	6763
410	JANESVILLE, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	51071	5142	7716	19292	39.97	2	7000
411	KENOSHA, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	77685	9028	7543	29411	38.50	1	8104

(continued on next page)

(continued from previous page)

412	MADISON, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	170616	14879	8012	68996	30.55	1	8674
413	MILWAUKEE, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	636212	79322	7028	253489	38.50	1	6977
414	RACINE, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	85725	10317	7588	32982	38.50	1	20900
415	WAUKESHA, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	50319	4232	7898	18347	38.50	1	3265
416	WAUWATOSA, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	51308	9221	10264	19613	38.50	1	5473
417	WEST ALLIS, WI	WI	NORTH CENTRAL	EAST NORTH CENTRAL	63982	9958	8311	26282	38.50	1	6063
418	CASPER, WY	WY	WEST	MOUNTAIN	51016	3722	9700	20259	27.15	2	6617

OBS	JANTEMP	JULYTEMP	RAINFALL	POPGRP	OVR65GRP	INCOMGRP	HOUSGRP	ELECGRP	CRIMEGRP	JANGRP	JULYGRP	RAINGRP
401	38.4	62.9	36.23	1	2	2	1	1	4	2	1	2
402	40.6	65.3	38.85	3	3	2	3	1	4	2	1	2
403	25.7	69.7	16.71	3	3	1	3	1	3	1	1	1
404	40.3	65.0	37.17	3	3	1	3	1	4	2	1	2
405	32.9	74.5	42.43	1	2	2	2	1	4	2	1	2
406	32.8	75.4	40.72	1	2	1	2	1	3	2	2	2
407	15.3	71.4	30.41	1	1	2	1	1	2	1	1	2
408	9.8	70.8	30.31	1	1	1	1	1	1	1	1	2
409	14.0	69.5	28.00	2	2	1	2	1	2	1	1	2
410	18.9	73.9	32.29	1	1	2	1	1	2	1	1	2
411	20.4	70.1	32.24	2	2	1	2	1	3	1	1	2
412	15.6	70.6	30.84	3	2	2	3	1	3	1	1	2
413	18.7	70.5	30.94	3	3	1	3	1	2	1	1	2
414	20.4	71.7	34.19	2	2	2	2	1	4	1	1	2
415	18.1	71.8	32.02	1	1	2	1	1	1	1	1	2
416	18.7	70.5	30.94	1	2	2	1	1	1	1	1	2
417	19.7	73.4	31.27	1	2	2	2	1	2	1	1	2
418	22.2	70.9	11.43	1	1	2	1	1	2	1	1	1

Descriptive statistics with FREQ, SUMMARY, and TABULATE The observations in data set CITIES are now ready to be used as input to the FREQ, SUMMARY, and TABULATE procedures. Since the SUMMARY procedure does not produce any printed output, the PRINT procedure is used to print the output data set from SUMMARY.

Comparing the output from the UNIVARIATE, FREQ, SUMMARY, and TABULATE procedures, the list of descriptive statistics available with each procedure is similar, but the procedures differ in other important respects. See the descriptions of these procedures for more detailed information.

```
PROC FREQ DATA=CITIES;
   TABLES DIVISION JANGRP JULYGRP POPGRP REGION INCOMGRP
      CRIMEGRP RAINGRP OVR65GRP HOUSGRP ELECGRP;
   TABLES POPGRP*CITYGOVT;
   TABLES REGION*OVR65GRP / NOCOL NOPERCENT;
   TABLES JANGRP*JULYGRP*ELECGRP / LIST;
   TABLES (POPGRP INCOMGRP)*CRIMEGRP / NOCOL NOPERCENT
      EXPECTED DEVIATION CELLCHI2 CHISQ;
   TABLES CITYGOVT*CRIMEGRP / ALL NOCOL NOPERCENT;
   FORMAT POPGRP PC. OVR65GRP OC. INCOMGRP IC. HOUSGRP HC.
      ELECGRP EC. CITYGOVT GC. CRIMEGRP CC. JANGRP JAN.
      JULYGRP JUL. RAINGRP RC.;
   TITLE 'CENSUS DATA: OUTPUT FROM FREQ PROCEDURE';
```

Output 10.4 Census Data: Output from PROC FREQ

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 27

DIVISION	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
EAST NORTH CENTR	83	19.9	83	19.9
EAST SOUTH CENTR	14	3.3	97	23.2
MIDDLE ATLANTIC	37	8.9	134	32.1
MOUNTAIN	28	6.7	162	38.8
NEW ENGLAND	37	8.9	199	47.6
PACIFIC	90	21.5	289	69.1
SOUTH ATLANTIC	49	11.7	338	80.9
WEST NORTH CENTR	29	6.9	367	87.8
WEST SOUTH CENTR	51	12.2	418	100.0

MEAN JANUARY TEMPERATURE (DEGREES F)

JANGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 32 DEGREES	208	49.8	208	49.8
32 DEGREES OR HI	210	50.2	418	100.0

MEAN JULY TEMPERATURE (DEGREES F)

JULYGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 75 DEGREES	241	57.7	241	57.7
75 DEGREES OR HI	177	42.3	418	100.0

1980 POPULATION ESTIMATE

POPGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
50,001-75,000	162	38.8	162	38.8
75,001-100,000	87	20.8	249	59.6
OVER 100,000	169	40.4	418	100.0

REGION	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
NORTH CENTRAL	112	26.8	112	26.8
NORTHEAST	74	17.7	186	44.5
SOUTH	114	27.3	300	71.8
WEST	118	28.2	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 28

PER CAPITA MONEY INCOME (1979)

INCOMGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
AVERAGE OR ABOVE	243	58.1	243	58.1
BELOW AVERAGE	175	41.9	418	100.0

CRIME RATE/100,000 POP (1981)

CRIMEGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
6000 OR LESS	109	26.1	109	26.1
6001-7500	102	24.4	211	50.5
7501-9500	107	25.6	318	76.1
OVER 9500	100	23.9	418	100.0

(continued on next page)

(continued from previous page)

ANNUAL PRECIPITATION (INCHES)

RAINGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LESS THAN 19 INC	99	23.7	99	23.7
19-44 INCHES	228	54.5	327	78.2
OVER 44 INCHES	91	21.8	418	100.0

PERSONS 65 & OVER (1980)

OVR65GRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
7000 OR LESS	111	26.6	111	26.6
7001-17000	209	50.0	320	76.6
OVER 17000	98	23.4	418	100.0

HOUSING UNITS (1980)

HOUSGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LOW	103	24.6	103	24.6
MEDIUM	212	50.7	315	75.4
HIGH	103	24.6	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 29

TYPICAL RESID. ELECTRIC BILL/MO. (1982)

ELECGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LOW	110	26.3	110	26.3
MEDIUM	241	57.7	351	84.0
HIGH	67	16.0	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 30

TABLE OF POPGRP BY CITYGOVT

POPGRP(1980 POPULATION ESTIMATE)
CITYGOVT(FORM OF CITY GOVERNMENT (1982))

FREQUENCY PERCENT ROW PCT COL PCT	MAYOR COUNCIL	COUNCIL MANAGER	COMMISSION	TOTAL
50,001-75,000	55 13.16 33.95 33.74	104 24.88 64.20 43.33	3 0.72 1.85 20.00	162 38.76
75,001-100,000	33 7.89 37.93 20.25	50 11.96 57.47 20.83	4 0.96 4.60 26.67	87 20.81
OVER 100,000	75 17.94 44.38 46.01	86 20.57 50.89 35.83	8 1.91 4.73 53.33	169 40.43
TOTAL	163 39.00	240 57.42	15 3.59	418 100.00

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 31

TABLE OF REGION BY OVR65GRP

REGION OVR65GRP(PERSONS 65 & OVER (1980))

FREQUENCY ROW PCT	7000 OR LESS	7001-17000	OVER 17000	TOTAL
NORTH CENTRAL	33 29.46	55 49.11	24 21.43	112
NORTHEAST	7 9.46	53 71.62	14 18.92	74
SOUTH	28 24.56	48 42.11	38 33.33	114
WEST	43 36.44	53 44.92	22 18.64	118
TOTAL	111	209	98	418

JANGRP	JULYGRP	ELECGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 32 DEGREES	UNDER 75 DEGREES	LOW	48	11.5	48	11.5
UNDER 32 DEGREES	UNDER 75 DEGREES	MEDIUM	70	16.7	118	28.2
UNDER 32 DEGREES	UNDER 75 DEGREES	HIGH	39	9.3	157	37.6
UNDER 32 DEGREES	75 DEGREES OR HI	LOW	15	3.6	172	41.1
UNDER 32 DEGREES	75 DEGREES OR HI	MEDIUM	22	5.3	194	46.4
UNDER 32 DEGREES	75 DEGREES OR HI	HIGH	14	3.3	208	49.8
32 DEGREES OR HI	UNDER 75 DEGREES	LOW	10	2.4	218	52.2
32 DEGREES OR HI	UNDER 75 DEGREES	MEDIUM	65	15.6	283	67.7
32 DEGREES OR HI	UNDER 75 DEGREES	HIGH	9	2.2	292	69.9
32 DEGREES OR HI	75 DEGREES OR HI	LOW	37	8.9	329	78.7
32 DEGREES OR HI	75 DEGREES OR HI	MEDIUM	84	20.1	413	98.8
32 DEGREES OR HI	75 DEGREES OR HI	HIGH	5	1.2	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 32

TABLE OF POPGRP BY CRIMEGRP

POPGRP(1980 POPULATION ESTIMATE) CRIMEGRP(CRIME RATE/100,000 POP (1981))

FREQUENCY EXPECTED DEVIATION CELL CHI2 ROW PCT	6000 OR LESS	6001-7500	7501-9500	OVER 9500	TOTAL
50,001-75,000	68 42.2 25.8 15.7033 41.98	47 39.5 7.5 1.41115 29.01	27 41.5 -14.5 5.04834 16.67	20 38.8 -18.8 9.07697 12.35	162
75,001-100,000	22 22.7 -0.7 0.02078 25.29	24 21.2 2.8 .361511 27.59	25 22.3 2.7 .334574 28.74	16 20.8 -4.8 1.11317 18.39	87
OVER 100,000	19 44.1 -25.1 14.261 11.24	31 41.2 -10.2 2.54229 18.34	55 43.3 11.7 3.18556 32.54	64 40.4 23.6 13.74 37.87	169
TOTAL	109	102	107	100	418

(continued on next page)

(continued from previous page)

STATISTICS FOR TABLE OF POPGRP BY CRIMEGRP

STATISTIC	DF	VALUE	PROB
CHI-SQUARE	6	66.799	0.000
LIKELIHOOD RATIO CHI-SQUARE	6	68.810	0.000
MANTEL-HAENSZEL CHI-SQUARE	1	63.089	0.000
PHI		0.400	
CONTINGENCY COEFFICIENT		0.371	
CRAMER'S V		0.283	

SAMPLE SIZE = 418

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 33

TABLE OF INCOMGRP BY CRIMEGRP

INCOMGRP(PER CAPITA MONEY INCOME (1979)) CRIMEGRP(CRIME RATE/100,000 POP (1981))

FREQUENCY EXPECTED DEVIATION CELL CHI2 ROW PCT	6000 OR LESS	6001-750 0	7501-950 0	OVER 950 0	TOTAL
AVERAGE OR ABOVE	51	59	62	71	243
	63.4	59.3	62.2	58.1	
	-12.4	-0.3	-0.2	12.9	
	2.41326	.001484	7E-04	2.84747	
	20.99	24.28	25.51	29.22	
BELOW AVERAGE	58	43	45	29	175
	45.6	42.7	44.8	41.9	
	12.4	0.3	0.2	-12.9	
	3.35098	.002061	9E-04	3.95391	
	33.14	24.57	25.71	16.57	
TOTAL	109	102	107	100	418

STATISTICS FOR TABLE OF INCOMGRP BY CRIMEGRP

STATISTIC	DF	VALUE	PROB
CHI-SQUARE	3	12.571	0.006
LIKELIHOOD RATIO CHI-SQUARE	3	12.770	0.005
MANTEL-HAENSZEL CHI-SQUARE	1	11.271	0.001
PHI		0.173	
CONTINGENCY COEFFICIENT		0.171	
CRAMER'S V		0.173	

SAMPLE SIZE = 418

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 34

TABLE OF CITYGOVT BY CRIMEGRP

CITYGOVT(FORM OF CITY GOVERNMENT (1982)) CRIMEGRP(CRIME RATE/100,000 POP (1981))

FREQUENCY ROW PCT	6000 OR LESS	6001-750 0	7501-950 0	OVER 950 0	TOTAL
MAYOR COUNCIL	42	44	39	38	163
	25.77	26.99	23.93	23.31	
COUNCIL MANAGER	65	55	62	58	240
	27.08	22.92	25.83	24.17	
COMMISSION	2	3	6	4	15
	13.33	20.00	40.00	26.67	
TOTAL	109	102	107	100	418

```
                CENSUS DATA: OUTPUT FROM FREQ PROCEDURE                    35

              STATISTICS FOR TABLE OF CITYGOVT BY CRIMEGRP


STATISTIC                     DF     VALUE       PROB
------------------------------------------------------------
CHI-SQUARE                     6     3.295       0.771
LIKELIHOOD RATIO CHI-SQUARE    6     3.320       0.768
MANTEL-HAENSZEL CHI-SQUARE     1     0.527       0.468
PHI                                  0.089
CONTINGENCY COEFFICIENT              0.088
CRAMER'S V                           0.063


STATISTIC                            VALUE        ASE
------------------------------------------------------------
GAMMA                                0.043       0.069
KENDALL'S TAU-B                      0.027       0.043
STUART'S TAU-C                       0.025       0.040

SOMERS' D C|R                        0.032       0.052
SOMERS' D R|C                        0.022       0.036

PEARSON CORRELATION                  0.036       0.048
SPEARMAN CORRELATION                 0.030       0.048

LAMBDA ASYMMETRIC C|R                0.019       0.031
LAMBDA ASYMMETRIC R|C                0.000       0.000
LAMBDA SYMMETRIC                     0.012       0.020

UNCERTAINTY COEFFICIENT C|R          0.003       0.003
UNCERTAINTY COEFFICIENT R|C          0.005       0.005
UNCERTAINTY COEFFICIENT SYM          0.004       0.004

SAMPLE SIZE = 418
WARNING:  33% OF THE CELLS HAVE EXPECTED COUNTS LESS
          THAN 5. CHI-SQUARE MAY NOT BE A VALID TEST.

ASE IS THE ASYMPTOTIC STANDARD ERROR.
R|C MEANS ROW VARIABLE DEPENDENT ON COLUMN VARIABLE.
```

```
                CENSUS DATA: OUTPUT FROM FREQ PROCEDURE                    36

              SUMMARY STATISTICS FOR CITYGOVT BY CRIMEGRP


  COCHRAN-MANTEL-HAENSZEL STATISTICS (BASED ON TABLE SCORES)

STATISTIC   ALTERNATIVE HYPOTHESIS    DF       VALUE      PROB
------------------------------------------------------------
    1       NONZERO CORRELATION       1       0.527     0.468
    2       ROW MEAN SCORES DIFFER    2       1.364     0.506
    3       GENERAL ASSOCIATION       6       3.287     0.772


TOTAL SAMPLE SIZE = 418
```

```
PROC SUMMARY DATA=CITIES;
   CLASS POPGRP REGION;
   VAR OVER65 INCOME HOUSING ELECTRIC CRIME;
   OUTPUT OUT=CITYDATA           N=NOVER65 NINCOME NHOUS NELEC NCRIME
                                 MEAN=
      STD(INCOME HOUSING ELECTRIC)=INCOMSTD HOUSESTD ELECSTD;
   FORMAT POPGRP PC.;

PROC PRINT DATA=CITYDATA;
   TITLE 'CENSUS DATA: RESULTS FROM SUMMARY PROCEDURE';
```

Output 10.5 Census Data: Results from PROC SUMMARY

CENSUS DATA: RESULTS FROM SUMMARY PROCEDURE 37

OBS	POPGRP	REGION	_TYPE_	_FREQ_	NOVER65	NINCOME	NHOUS	NELEC	NCRIME	OVER65	INCOME	HOUSING	ELECTRIC	CRIME	INCOMSTD	HOUSESTD	ELECSTD
1	.		0	418	418	418	418	418	418	20552.9	7569.34	71406	47.8885	7856.02	1627.25	177443	11.7452
2	.	NORTH CENTRAL	1	112	112	112	112	112	112	18935.4	7853.63	64422	43.3684	7525.53	1551.28	124931	7.0366
3	.	NORTHEAST	1	74	74	74	74	74	74	30571.3	6709.34	91026	58.1432	7129.92	1285.77	346647	11.5923
4	.	SOUTH	1	114	114	114	114	114	114	19860.9	7098.77	73108	43.5652	8536.21	1250.58	88274	7.7512
5	.	WEST	1	118	118	118	118	118	118	16474.1	8293.46	64085	49.9245	7967.92	1822.84	120687	13.9414
6	50,001-75,000		2	162	162	162	162	162	162	7044.2	7778.08	23675	47.9718	6737.57	1890.68	4024	11.8870
7	75,001-100,000		2	87	87	87	87	87	87	9843.2	7483.18	32799	49.6853	7667.21	1705.08	5872	9.8244
8	OVER 100,000		2	169	169	169	169	169	169	39015.5	7413.60	137034	46.8836	9025.34	1260.27	266117	12.4458
9	50,001-75,000	NORTH CENTRAL	3	46	46	46	46	46	46	6482.2	8255.61	22567	43.8065	5989.65	1863.10	3403	6.5461
10	50,001-75,000	NORTHEAST	3	33	33	33	33	33	33	8785.5	6903.67	24131	57.6345	5841.85	1044.48	3513	12.1465
11	50,001-75,000	SOUTH	3	40	40	40	40	40	40	7386.1	7144.55	24738	42.4635	8091.20	1263.73	3712	8.7433
12	50,001-75,000	WEST	3	43	43	43	43	43	43	5990.9	8527.63	23524	50.1360	6965.88	2421.11	4986	13.6489
13	75,001-100,000	NORTH CENTRAL	3	27	27	27	27	27	27	9605.7	7889.59	32066	43.1026	8012.70	1520.11	4248	7.0382
14	75,001-100,000	NORTHEAST	3	18	18	18	18	18	18	12179.1	6652.89	34101	56.4678	6976.72	1743.41	3246	7.9960
15	75,001-100,000	SOUTH	3	13	13	13	13	13	13	11785.0	6621.85	34460	44.7631	8311.46	1548.09	10536	6.0896
16	75,001-100,000	WEST	3	29	29	29	29	29	29	7743.8	8006.28	31927	53.8107	7485.31	1637.41	5611	9.7728
17	OVER 100,000	NORTH CENTRAL	3	39	39	39	39	39	39	40082.7	7354.59	136191	43.0356	8999.79	941.13	193427	7.7200
18	OVER 100,000	NORTHEAST	3	23	23	23	23	23	23	76223.2	6474.70	231557	60.1843	9097.91	1197.22	607198	13.2233
19	OVER 100,000	SOUTH	3	61	61	61	61	61	61	29762.1	7170.39	113063	44.0323	8875.92	1171.39	105536	7.3962
20	OVER 100,000	WEST	3	46	46	46	46	46	46	31777.5	8255.61	122274	47.2767	9208.85	1180.18	179241	15.9719

```
PROC TABULATE DATA=CITIES;
   TITLE 'CENSUS DATA: OUTPUT FROM TABULATE PROCEDURE';
   FORMAT POPGRP PC.;
   VAR INCOME;
   CLASS POPGRP REGION;
   TABLE POPGRP*REGION, INCOME*(N*F=4.0 MEAN STD);
```

Output 10.6 Census Data: Output from PROC TABULATE

CENSUS DATA: OUTPUT FROM TABULATE PROCEDURE 38

		PER CAPITA MONEY INCOME (1979)		
		N	MEAN	STD
1980 POPULATION ESTIMATE	REGION			
50,001-75,000	NORTH CENTRAL	46	8255.61	1863.10
	NORTHEAST	33	6903.67	1044.48
	SOUTH	40	7144.55	1263.73
	WEST	43	8527.63	2421.11
75,001-100,000	REGION			
	NORTH CENTRAL	27	7889.59	1520.11
	NORTHEAST	18	6652.89	1743.41
	SOUTH	13	6621.85	1548.09
	WEST	29	8006.28	1637.41

(continued on next page)

(continued from previous page)

OVER 100,000	REGION			
	NORTH CENTRAL	39	7354.59	941.13
	NORTHEAST	23	6474.70	1197.22
	SOUTH	61	7170.39	1171.39
	WEST	46	8255.61	1180.18

Sales Data: Example 2

This example illustrates some advanced INPUT features and displays sales for three products using the SUMMARY, PRINT, and CHART procedures.

Using FORMAT The input data contain sales for each product recorded monthly. In the first step in the example, the FORMAT procedure is used to expand the monthly codes 1-12 into abbreviations.

```
PROC FORMAT;
   VALUE MMM 1='JAN' 2='FEB' 3='MAR' 4='APR' 5='MAY' 6='JUN'
             7='JUL' 8='AUG' 9='SEP' 10='OCT' 11='NOV' 12='DEC';
```

Data set SALES The statements below create a SAS data set SALES.

The INPUT statement reads in a hierarchical file containing three different types of records. This hierarchical arrangement reduces the number of keystrokes required during data entry as well as the size of the file. The types of input records are

- product records with a PRODUCT number
- date records with a DATE value
- sales records with a variable number of CUSTOMER codes and UNIT values.

Each record starts with a record TYPE coded P, D, or S read with the trailing @ format modifier. This "held" input method allows SAS to read further fields from the same INPUT statement in different forms depending on the record's TYPE. P and D input records are retained with a RETAIN statement. For S records, each pair of CUSTOMER and UNIT is output. The data are coded so that a CUSTOMER value of $$ signifies the end of the record. The trailing @ format modifier is discussed in the **INPUT Statement** description in "Statements Used in the DATA Step." (The technique for hierarchical data is discussed in greater detail in the *SAS Applications Guide*.)

```
DATA SALES;
   TITLE 'RESULTS OF SALES DATA';
   LENGTH PRODUCT $8 CUSTOMER $8;
   RETAIN PRODUCT MONTH YEAR DATE;
   FORMAT DATE DATE. MONTH MMM.;
   INPUT TYPE $ @;
   DROP TYPE;
   IF TYPE = 'P' THEN INPUT PRODUCT @;
   ELSE IF TYPE = 'D' THEN DO;
      INPUT DATE : MONYY5.@;
      MONTH=MONTH(DATE);
      YEAR=YEAR(DATE);
      END;
```

```
   ELSE IF TYPE = 'S' THEN DO WHILE(1);
      INPUT CUSTOMER $ @;
      IF CUSTOMER='$$' THEN RETURN;
      INPUT UNITS @;
      OUTPUT;
      END;
CARDS;
P 8401
D DEC81
S GR 1 JP 11 JP 19 OT 1 OR 8 JP 20 GU 5 $$
D NOV81
S GR 1 GR 1 JP 20 GR 1 JP 20 GR 1 $$
D OCT81
S GE 15 OR 11 JP 20 TU 4 GR 1 GR 2 JP 20 GR 1 GR 1 JP 20 GR 1 $$
D SEP81
S JP 20 GR 1 TU 3 GE 15 TU 3 GR 1 JP 28 GU 5 JP 2 OR 15 GR 1 GE 15 $$
D AUG81
S JP 30 TU 3 GR 1 OR 15 GU 3 JP 20 GE 15 GR 2 JP 20 GR 1 $$
D JUL81
S JP 20 GE 15 OR 15 GR 1 GU 5 JP 15 JP 15 TU 3 OR 11 JP 20 GU 5 GE 15
  OR 11 $$
D JUN81
S GE 11 JP 20 OR 10 TU 3 GR 1 GU 10 GU 5 GR 2 OT 1 JP 20 OR 5 GR 1
  GE 10 OR 10 JP 3 GU 4 JP 17 GR 1 $$
D MAY81
S TU 5 OR 2 GU 5 OR 9 JP 20 GR 1 OR 10 OT 1 GR 1 GR 1 JP 12 JP 8 OR 10
  GR 2 $$
D APR81
S GR 2 GU 5 OR 10 GR 1 GR 1 JP 20 OR 10 JP 10 GR 2 GU 5 GR 1 OR 5 GR 1
  GR 1 GR 2 TU 5 OR 10 $$
D MAR81
S JP 15 GR 1 GR 1 GR 1 GR 2 OR 11 JP 12 GU 5 JP 3 TU 5 OT 1 OR 5 JP 15
  OR 10 $$
D FEB81
S JP 5 JP 10 GU 5 JP 10 OR 10 JP 10 OR 10 $$
D JAN81
S JP 10 OR 10 JP 15 OR 10 GU 5 OR 8 TU 5 $$
D DEC80
S JP 10 JP 10 OR 10 JP 10 OR 10 GU 5 $$
D NOV80
S TU 3 JP 10 OR 5 OT 1 JP 1 JP 9 $$
D OCT80
S JP 10 OT 1 JP 5 OR 10 JP 10 OR 9 TU 3 TU 1 $$
D SEP80
S JP 10 OR 10 JP 10 OR 10 JP 10 OR 5 GU 5 OR 2 JP 10 $$
D AUG80
S JP 10 OR 10 JP 10 OR 9 OR 1 TU 6 $$
D JUL80
S GU 5 JP 10 OR 10 JP 5 TU 5 OR 10 JP 8 JP 7 $$
D JUN80
S TU 5 JP 9 GU 5 JP 1 OR 10 JP 10 OR 10 $$
D MAY80
S JP 10 TU 4 TU 4 OR 10 JP 10 OR 10 JP 10 GU 5 $$
D APR80
S OR 10 JP 10 OT 1 OT 1 OR 10 GU 6 JP 10 OR 7 OR 3 JP 10 $$
```

```
D MAR80
S OR 11 JP 10 JP 10 OR 5 JP 10 GU 4 JP 10 $$
D FEB80
S JP 10 OR 9 OR 5 JP 10 JP 5 $$
D JAN80
S OR 11 JP 10 JP 10 OR 5 JP 10 GU 4 JP 10 $$
D DEC79
S OR 5 TU 10 JP 5 OR 5 JP 10 $$
D NOV79
S JP 5 OR 5 GU 5 JP 5 JP 1 JP 1 JP 5 $$
D OCT79
S OR 11 TU 10 JP 5 GU 4 JP 5 JP 5 OR 10 JP 5 OR 10 JP 10 $$
D SEP79
S OT 1 OR 10 GU 5 JP 10 OR 12 JP 10 JP 10 OR 10 $$
D AUG79
S GU 4 OR 12 JP 10 JP 10 JP 5 GU 4 OR 10 JP 5 JP 5 $$
D JUL79
S OR 10 JP 5 OT 2 JP 5 GU 4 JP 5 OR 10 JP 3 $$
P 9054
D DEC81
S OT 2 AM 4 $$
D NOV81
S DR 1 DR 4 $$
D OCT81
S CH 3 GR 1 OT 1 CH 3 OT 1 LU 1 $$
D SEP81
S CH 3 OT 1 OT 1 CH 3 DR 5 OT 1 CA 1 $$
D AUG81
S OT 4 OR 1 $$
D JUL81
S DR 3 OR 2 CH 3 OR 3 OT 1 CA 2 OT 1 CA 2 OT 1 OT 1 $$
D JUN81
S AM 4 DR 4 OT 3 OR 2 CH 3 OT 1 OT 1 DR 3 OR 2 AM 4 DR 3 $$
D MAY81
S DR 1 GA 2 CA 2 OR 1 DR 4 OR 3 $$
D APR81
S AM 4 OT 1 GA 1 DR 1 CH 3 GA 1 DR 2 OT 1 AM 4 OR 2 $$
D MAR81
S AM 4 CA 2 DR 2 DR 2 GA 2 AM 4 OR 2 DR 1 OR 1 AM 1 CH 3 AM 3 $$
D FEB81
S AM 4 DR 2 CA 2 CH 3 AM 3 CA 2 GA 2 $$
D JAN81
S AM 4 DR 2 DR 2 CA 2 DR 2 DR 1 $$
D DEC80
S GA 2 AM 4 CA 2 AM 4 CH 2 AM 4 GA 3 $$
D NOV80
S GA 1 GA 2 AM 4 DR 1 $$
D OCT80
S AM 1 AM 3 GA 2 CH 2 OR 2 DR 2 AM 4 DR 2 OR 1 $$
D SEP80
S AM 4 DR 1 CH 1 DR 1 AM 4 GA 2 DR 1 $$
D AUG80
S GA 2 AM 2 CH 1 AM 2 CH 2 CH 3 DR 2 AM 2 AM 2 $$
D JUL80
S GA 2 AM 4 DR 1 GA 2 AM 4 GA 1 AM 1 AM 3 CH 10 $$
```

```
D JUN80
S DR 1 AM 2 GA 1 DR 2 AM 2 CA 3 CA 2 $$
D MAY80
S GA 2 CH 10 OT 1 CA 3 CH 10 CA 3 $$
D APR80
S CH 10 DR 2 CH 10 OT 1 AM 4 DR 2 CA 3 CH 10 OT 1 CA 3 $$
D MAR80
S AM 4 DR 1 CA 4 CH 10 DR 2 GA 2 CH 10 CA 4 OT 1 $$
D FEB80
S CA 4 CH 10 AM 3 CA 4 CA 4 DR 2 CH 8 OT 1 OT 10 DR 1 CH 6 $$
D JAN80
S AM 2 CA 3 CH 6 GA 1 CH 4 OT 1 CA 3 OT 5 CH 4 OT 5 OT 1 CH 6 DR 2 CA 4
  $$
D DEC79
S OT 2 AM 1 CH 4 CA 4 $$
D NOV79
S CH 4 CA 4 DR 1 OT 5 CA 4 OT 2 DR 2 $$
D OCT79
S OT 4 CA 3 CA 5 DR 1 CH 4 CA 5 $$
D SEP79
S OT 5 DR 1 CA 2 CH 2 CA 3 OT 4 CA 2 DR 1 CA 3 $$
D AUG79
S DR 1 OT 4 CA 2 CA 5 DR 1 CA 5 CA 4 $$
D JUL79
S OT 4 CH 4 OT 1 CA 3 CA 5 DR 1 CH 3 CH 3 CH 4 $$
D JUN79
S AM 1 CH 6 DR 1 CA 3 CH 6 CA 2 DR 1 $$
D MAY79
S CH 4 OT 4 CA 3 CH 6 DR 1 $$
D APR79
S OT 1 CH 6 DR 1 OT 4 CA 5 CH 4 DR 1 CH 4 $$
D MAR79
S OT 4 GA 1 CH 4 CH 4 DR 1 OT 4 CH 4 $$
D FEB79
S DR 1 OT 2 OT 4 GA 1 CH 4 $$
D JAN79
S OT 4 CA 3 OT 4 $$
P 4005
D DEC81
S I 6 B 13 AM 7 DT 12 O 2 OT 3 $$
D NOV81
S OT 5 DT 29 DH 2 I 7 B 14 AM 7 $$
D OCT81
S B 23 DT 31 I 6 AM 7 OT 7 DH 2 O 5 $$
D SEP81
S B 24 DT 26 I 5 O 10 AM 8 OT 6 DH 3 $$
D AUG81
S B 31 DT 20 OT 17 I 5 DH 3 O 5 AM 6 $$
D JUL81
S B 24 DT 22 I 5 DH 4 OT 12 O 10 AM 7 $$
D JUN81
S B 20 DT 6 I 4 O 10 OT 6 AM 7 DH 3 $$
D MAY81
S B 30 DT 20 I 5 O 5 AM 8 DH 2 OT 11 $$
D APR81
```

```
S B 22 DT 12 O 15 I 4 AM 7 DH 3 OT 7 $$
D MAR81
S B 29 DT 15 O 20 I 4 DH 5 AM 8 OT 5 $$
D FEB81
S B 23 DT 8 O 10 I 3 AM 6 OT 5 DH 2 $$
D JAN81
S B 25 DT 15 O 10 AM 7 DH 6 OT 4 $$
D DEC80
S B 20 DT 7 O 5 AM 7 DH 1 OT 2 $$
D NOV80
S B 16 DT 15 O 10 AM 7 DH 5 OT 3 $$
D OCT80
S B 15 DT 20 AM 6 O 10 DH 3 OT 4 $$
D SEP80
S B 11 DT 8 AM 7 O 15 DH 3 OT 4 $$
D AUG80
S B 16 DT 0 AM 7 O 5 DH 2 OT 2 $$
D JUL80
S B 15 DT 0 AM 7 O 15 DH 3 OT 2 $$
D JUN80
S B 19 DT 0 AM 8 O 5 DH 7 OT 3 $$
D MAY80
S B 20 DT 4 AM 8 O 5 DH 3 OT 3 $$
D APR80
S B 22 DT 13 AM 8 O 10 DH 3 OT 3 $$
D MAR80
S B 20 DT 12 AM 9 O 15 DH 2 OT 5 $$
D FEB80
S B 21 DT 7 AM 7 O 10 DH 2 OT 2 $$
D JAN80
S B 19 DT 0 AM 8 DH 3 OT 7 $$
D DEC79
S B 10 DT 4 AM 7 DH 3 OT 5 $$
D NOV79
S B 19 DT 12 AM 6 DH 2 OT 2 $$
D OCT79
S B 24 DT 17 AM 6 DH 0 OT 1 $$
D SEP79
S B 15 DT 10 AM 4 DH 2 OT 3 $$
D AUG79
S B 16 DT 12 AM 8 DH 4 OT 2 $$
D JUL79
S B 17 DT 8 AM 6 DH 2 OT 3 $$
D JUN79
S B 18 DT 11 AM 2 DH 2 OT 2 $$
D MAY79
S B 11 DT 19 AM 8 DH 3 OT 8 $$
D APR79
S B 14 DT 14 AM 8 DH 0 OT 15 $$
D MAR79
S B 12 DT 14 AM 6 DH 2 OT 26 $$
D FEB79
S B 15 DT 11 AM 3 DH 0 OT 28 $$
D JAN79
S B 15 DT 8 AM 4 DH 3 OT 26 $$ ;
```

Data set SALES2 The next DATA step shows how to use a SET statement with a KEEP (or DROP) statement to subset the variables in an existing SAS data set. These statements:

```
DATA SALES2;
   SET SALES;
   KEEP PRODUCT DATE YEAR UNITS;
```

create data set SALES2, which contains only the variables PRODUCT, DATE, YEAR, and UNITS from data set SALES.

Using SUMMARY and PRINT Below are the statements that ask PROC SUMMARY to create an output data set containing summary statistics for two CLASS variables, PRODUCT and DATE. Since SUMMARY produces no printed output, the PRINT procedure is used to display the contents of the SUMMARY data set.

```
PROC SUMMARY DATA=SALES;
   CLASS PRODUCT DATE;
   VAR UNITS;
   ID YEAR;
   OUTPUT OUT=B SUM=SALES;
PROC PRINT DATA=B;
```

Output 10.7 Results of Sales Data

RESULTS OF SALES DATA 1

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
1	1981		.	0	742	4781
2	1979		01JAN79	1	8	67
3	1979		01FEB79	1	10	69
4	1979		01MAR79	1	12	82
5	1979		01APR79	1	13	77
6	1979		01MAY79	1	10	67
7	1979		01JUN79	1	12	55
8	1979		01JUL79	1	22	108
9	1979		01AUG79	1	21	129
10	1979		01SEP79	1	22	125
11	1979		01OCT79	1	21	145
12	1979		01NOV79	1	19	90
13	1979		01DEC79	1	14	75
14	1980		01JAN80	1	26	144
15	1980		01FEB80	1	22	141
16	1980		01MAR80	1	22	161
17	1980		01APR80	1	26	173
18	1980		01MAY80	1	20	135
19	1980		01JUN80	1	20	105
20	1980		01JUL80	1	23	130
21	1980		01AUG80	1	21	96
22	1980		01SEP80	1	22	134
23	1980		01OCT80	1	23	126
24	1980		01NOV80	1	16	93
25	1980		01DEC80	1	19	118
26	1981		01JAN81	1	19	143
27	1981		01FEB81	1	21	135
28	1981		01MAR81	1	33	200
29	1981		01APR81	1	34	181
30	1981		01MAY81	1	27	181
31	1981		01JUN81	1	36	220
32	1981		01JUL81	1	30	254
33	1981		01AUG81	1	19	202
34	1981		01SEP81	1	26	206
35	1981		01OCT81	1	24	187
36	1981		01NOV81	1	14	113
37	1981		01DEC81	1	15	114
38	1981	4005	.	2	212	1967

(continued on next page)

(continued from previous page)

39	1981	8401	.	2	270	2062
40	1981	9054	.	2	260	752
41	1979	4005	01JAN79	3	5	56
42	1979	4005	01FEB79	3	5	57
43	1979	4005	01MAR79	3	5	60
44	1979	4005	01APR79	3	5	51
45	1979	4005	01MAY79	3	5	49
46	1979	4005	01JUN79	3	5	35
47	1979	4005	01JUL79	3	5	36
48	1979	4005	01AUG79	3	5	42
49	1979	4005	01SEP79	3	5	34
50	1979	4005	01OCT79	3	5	48
51	1979	4005	01NOV79	3	5	41
52	1979	4005	01DEC79	3	5	29
53	1980	4005	01JAN80	3	5	37
54	1980	4005	01FEB80	3	6	49

RESULTS OF SALES DATA 2

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
55	1980	4005	01MAR80	3	6	63
56	1980	4005	01APR80	3	6	59
57	1980	4005	01MAY80	3	6	43
58	1980	4005	01JUN80	3	6	42
59	1980	4005	01JUL80	3	6	42
60	1980	4005	01AUG80	3	6	32
61	1980	4005	01SEP80	3	6	48
62	1980	4005	01OCT80	3	6	58
63	1980	4005	01NOV80	3	6	56
64	1980	4005	01DEC80	3	6	42
65	1981	4005	01JAN81	3	6	67
66	1981	4005	01FEB81	3	7	57
67	1981	4005	01MAR81	3	7	86
68	1981	4005	01APR81	3	7	70
69	1981	4005	01MAY81	3	7	81
70	1981	4005	01JUN81	3	7	56
71	1981	4005	01JUL81	3	7	84
72	1981	4005	01AUG81	3	7	87
73	1981	4005	01SEP81	3	7	82
74	1981	4005	01OCT81	3	7	81
75	1981	4005	01NOV81	3	6	64
76	1981	4005	01DEC81	3	6	43
77	1979	8401	01JUL79	3	8	44
78	1979	8401	01AUG79	3	9	65
79	1979	8401	01SEP79	3	8	68
80	1979	8401	01OCT79	3	10	75
81	1979	8401	01NOV79	3	7	27
82	1979	8401	01DEC79	3	5	35
83	1980	8401	01JAN80	3	7	60
84	1980	8401	01FEB80	3	5	39
85	1980	8401	01MAR80	3	7	60
86	1980	8401	01APR80	3	10	68
87	1980	8401	01MAY80	3	8	63
88	1980	8401	01JUN80	3	7	50
89	1980	8401	01JUL80	3	8	60
90	1980	8401	01AUG80	3	6	46
91	1980	8401	01SEP80	3	9	72
92	1980	8401	01OCT80	3	8	49
93	1980	8401	01NOV80	3	6	29
94	1980	8401	01DEC80	3	6	55
95	1981	8401	01JAN81	3	7	63
96	1981	8401	01FEB81	3	7	60
97	1981	8401	01MAR81	3	14	87
98	1981	8401	01APR81	3	17	91
99	1981	8401	01MAY81	3	14	87
100	1981	8401	01JUN81	3	18	134
101	1981	8401	01JUL81	3	13	151
102	1981	8401	01AUG81	3	10	110
103	1981	8401	01SEP81	3	12	109
104	1981	8401	01OCT81	3	11	96
105	1981	8401	01NOV81	3	6	44
106	1981	8401	01DEC81	3	7	65
107	1979	9054	01JAN79	3	3	11
108	1979	9054	01FEB79	3	5	12

RESULTS OF SALES DATA 3

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
109	1979	9054	01MAR79	3	7	22
110	1979	9054	01APR79	3	8	26
111	1979	9054	01MAY79	3	5	18
112	1979	9054	01JUN79	3	7	20
113	1979	9054	01JUL79	3	9	28
114	1979	9054	01AUG79	3	7	22
115	1979	9054	01SEP79	3	9	23
116	1979	9054	01OCT79	3	6	22
117	1979	9054	01NOV79	3	7	22
118	1979	9054	01DEC79	3	4	11
119	1980	9054	01JAN80	3	14	47
120	1980	9054	01FEB80	3	11	53
121	1980	9054	01MAR80	3	9	38
122	1980	9054	01APR80	3	10	46
123	1980	9054	01MAY80	3	6	29
124	1980	9054	01JUN80	3	7	13
125	1980	9054	01JUL80	3	9	28
126	1980	9054	01AUG80	3	9	18
127	1980	9054	01SEP80	3	7	14
128	1980	9054	01OCT80	3	9	19
129	1980	9054	01NOV80	3	4	8
130	1980	9054	01DEC80	3	7	21
131	1981	9054	01JAN81	3	6	13
132	1981	9054	01FEB81	3	7	18
133	1981	9054	01MAR81	3	12	27
134	1981	9054	01APR81	3	10	20
135	1981	9054	01MAY81	3	6	13
136	1981	9054	01JUN81	3	11	30
137	1981	9054	01JUL81	3	10	19
138	1981	9054	01AUG81	3	2	5
139	1981	9054	01SEP81	3	7	15
140	1981	9054	01OCT81	3	6	10
141	1981	9054	01NOV81	3	2	5
142	1981	9054	01DEC81	3	2	6

Data set C The purpose of the next DATA step is to reshape SUMMARY's output data set B for display with the CHART procedure. This example of the SET statement subsets the observations in data set B with a subsetting IF statement. The statements are

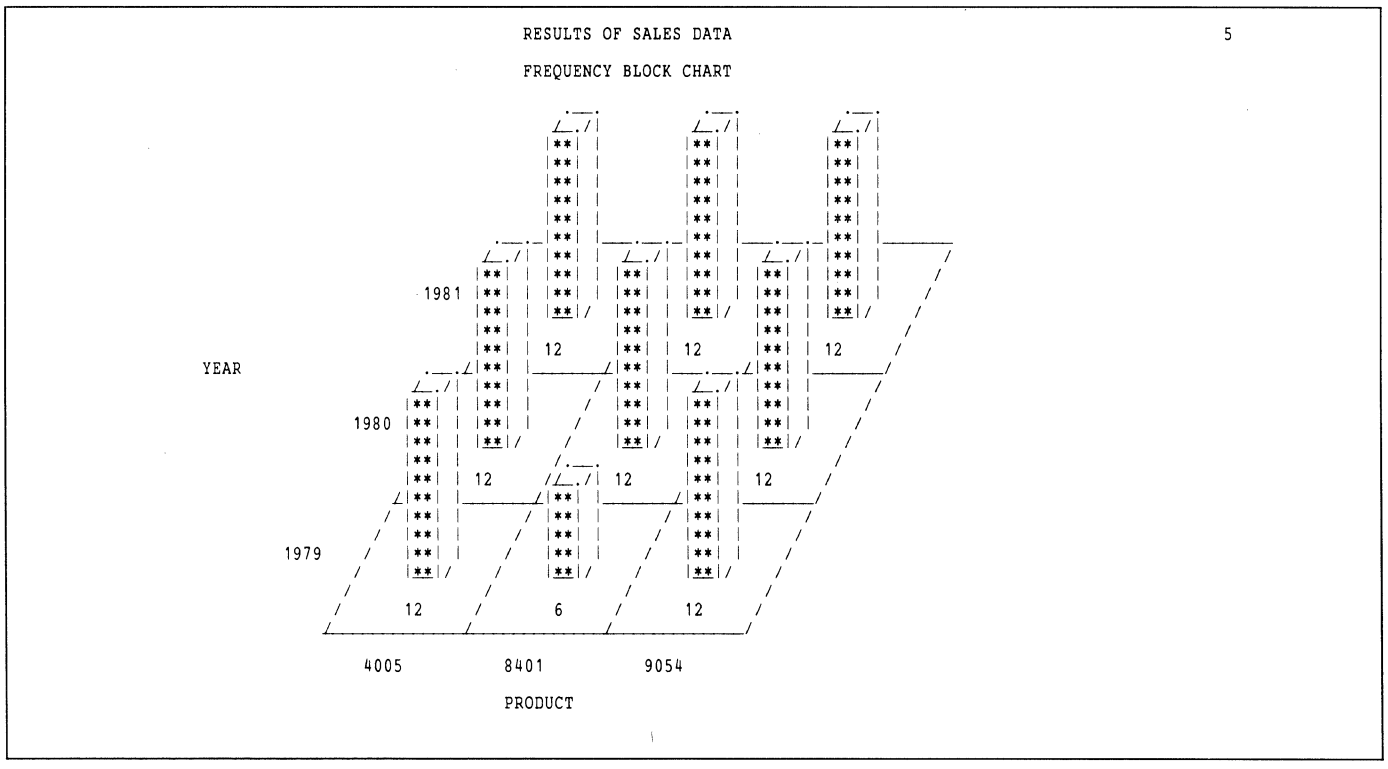
```
DATA C;
   SET B;
   IF _TYPE_=3;
```

Using CHART The statements below display the _TYPE_=3 observations from the SUMMARY data set, now in data set C, with PROC CHART:

```
PROC CHART;
   HBAR DATE / SUMVAR=SALES GROUP=PRODUCT;
   BLOCK PRODUCT / GROUP=YEAR;
   PIE PRODUCT / FREQ=SALES;
   VBAR YEAR / DISCRETE SUBGROUP=PRODUCT;
```

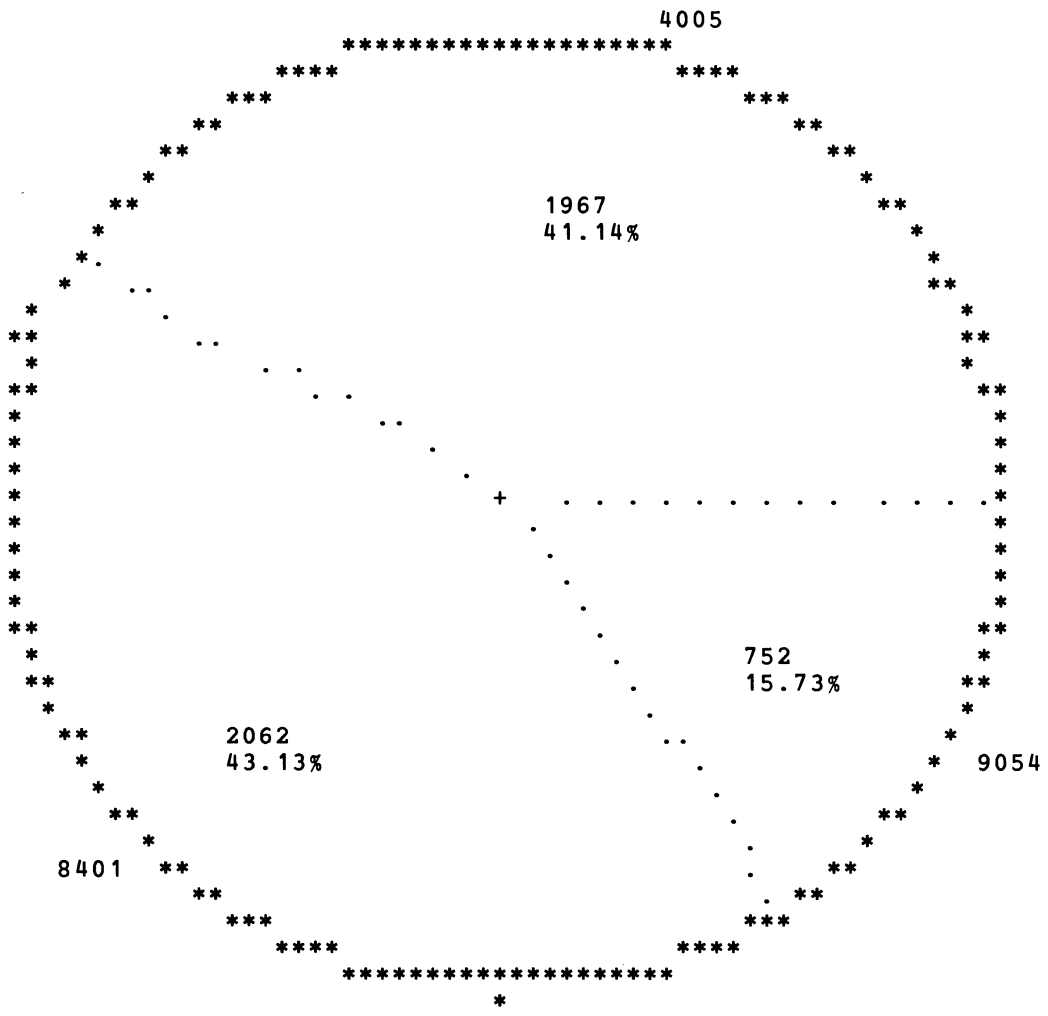
Output 10.8 Results of Sales Data Using PROC CHART

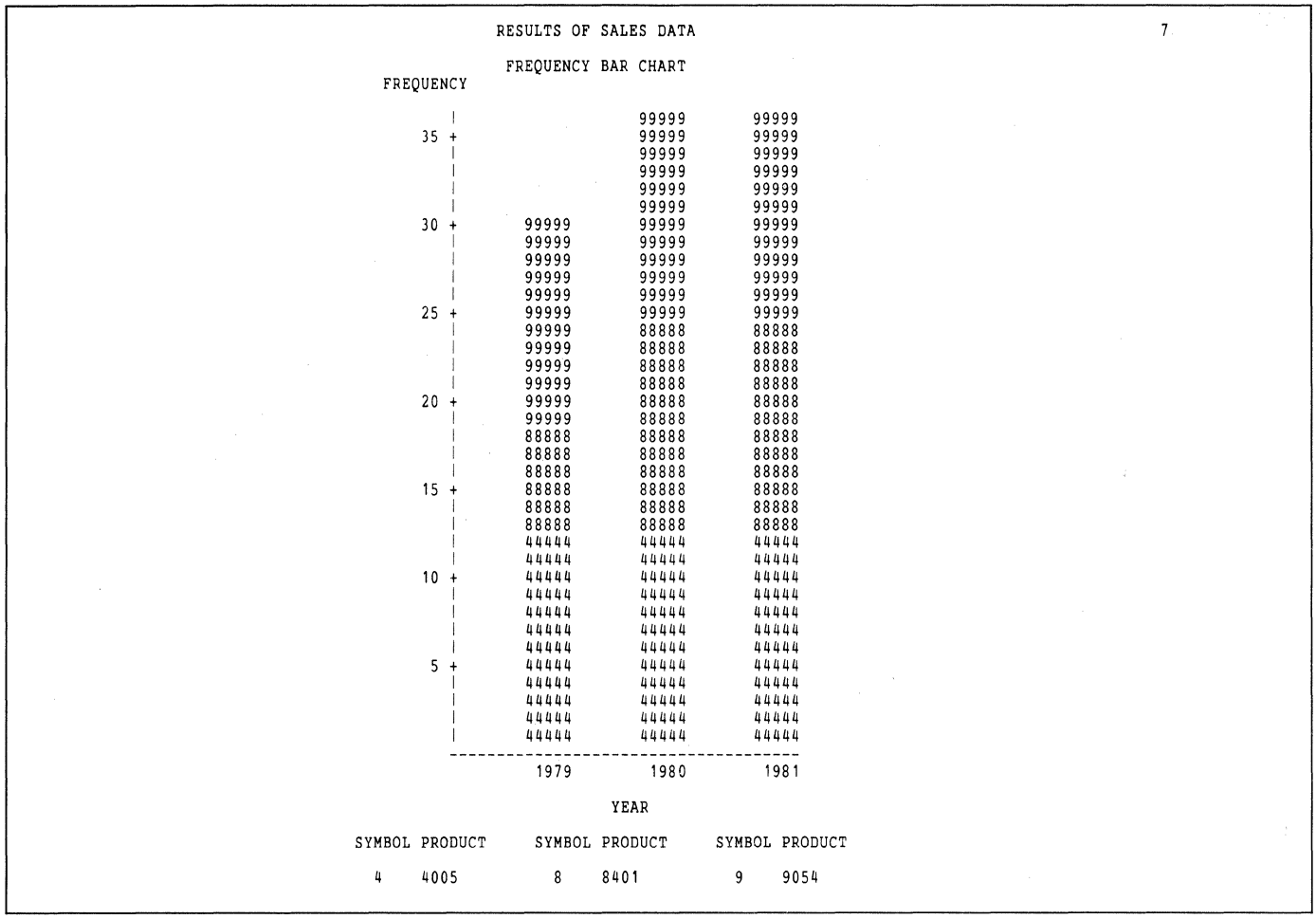
```
                                RESULTS OF SALES DATA                                    4

                                  BAR CHART OF SUMS

PRODUCT    MIDPOINT                                                          FREQ   SALES SUM
                    |
4005       10APR79  |*******************************                            6    308.0000
           17SEP79  |************************                                   6    230.0000
           24FEB80  |*************************                                  5    251.0000
           02AUG80  |**********************                                     5    222.0000
           09JAN81  |*******************************                            5    308.0000
           18JUN81  |************************************************           6    460.0000
           25NOV81  |******************                                         3    188.0000
                    |
8401       10APR79  |                                                           0      0.0000
           17SEP79  |*******************************                            6    314.0000
           24FEB80  |*****************************                              5    290.0000
           02AUG80  |****************************                               5    277.0000
           09JAN81  |*****************************                              5    294.0000
           18JUN81  |************************************************************ 6    682.0000
           25NOV81  |********************                                       3    205.0000
                    |
9054       10APR79  |***********                                                6    109.0000
           17SEP79  |*************                                              6    128.0000
           24FEB80  |**********************                                     5    213.0000
           02AUG80  |*********                                                  5     92.0000
           09JAN81  |*********                                                  5     87.0000
           18JUN81  |**********                                                 6    102.0000
           25NOV81  |**                                                         3     21.0000
                    |
                    ----+----+----+----+----+----+----+----+----+----+----+----+----+---
                        50   100  150  200  250  300  350  400  450  500  550  600  650

                                        SALES SUM
```



RESULTS OF SALES DATA 6

FREQ PIE CHART OF PRODUCT





See the CHART procedure for a complete discussion of HBAR, VBAR, BLOCK, and PIE charts.

REFERENCES

County and City Data Book, 1983 Files on Tape (machine-readable data file)/ prepared by the Bureau of the Census. Washington: The Bureau (producer and distributor), 1984.

County and City Data Book, 1983 Files on Tape Technical Documentation/ prepared by the Data User Services Division, Bureau of the Census. Washington: The Bureau, 1984.

FEATURES FOR BOTH DATA AND PROC STEPS

Introduction to the SAS® System

SAS® Statements Used Anywhere

SAS® Log and Procedure Output

SAS® Display Manager

SAS® Informats and Formats

Missing Values

SAS® Files

SAS® System Options

SAS® Macro Language

Chapter 11

Introduction to the SAS® System

SYSTEM DESIGN

Introduction

To provide insight into SAS System structure for the advanced programmer, this chapter describes the current internal organization of SAS as implemented for IBM System/360 and System/370-compatible machines. In its general charter to be a complete tool for managing and analyzing data, the design goals described below were adopted to make SAS easy to use. You do not need to learn any of the material in this section in order to use SAS. Other versions of SAS may look completely different internally yet perform the same from a user's point of view.

Design Goals

The following goals shaped the development of the SAS System:

- SAS software should be immediately usable, even by those who do not have extensive computer experience. Basics should be teachable in a few minutes. SAS should enable end-users to do their own computer processing if they choose to do so, rather than relying on computer specialists.
- SAS software should be a complete tool for the researcher; SAS should manage the data, as well as analyze them. Users should not have to learn one system to maintain and process data and another system to analyze them.
- Data management should be separate from analysis. Data processing functions need the flexibility of a programming language. Data analysis, however, can be packaged into the terms of the analysis. Thus, in the DATA step you specify a program for how to process data. In a PROC

step you simply request what you want done. Since all the data preparation is done in the DATA step, packaging the data in a uniform way, the procedure is freed from considering the form of the data.
- A special-purpose analysis that is not encompassed within any procedure that SAS provides can be programmed in the DATA step or with SAS/IML software or by combining existing PROCs and DATA step programming.
- SAS software should maintain a uniform and concise syntax. Procedures have similar control languages. The DATA step language is reasonably small. All control language is specified in a free-format syntax.
- SAS software should require a minimum of instructions. The user should not have to specify anything the system can figure out for itself. For example, most SAS DATA steps do not need declarations or attribute statements for variables or files. Where the user can make a choice, reasonable default choices are made by the system if the user does not specify.
- User errors should be diagnosed clearly and immediately.
- SAS software should not be limited in the number or size of data sets, nor in the number or size of the analyses.
- Results from one step should be available as input for the next step.
- Procedures should be as general and widely applicable as possible and have a variety of options for specifying the format, nature, and extent of their output.
- Procedures should be able to run repeatedly on breakdowns of the data according to a grouping variable. The BY-group facility handles this.
- Data should be handled in a simple, standardized, easy-to-use form. The rectangular self-describing data model serves most data processing needs well. Variables have few attributes, and they usually can be determined from context. Once the data are in the form of a SAS data set, the data are described inside the file.
- Missing values should be handled naturally and consistently.
- SAS software should operate interactively in the same form as in batch.
- SAS software should be extendable by users and shall be extended by SAS Institute.

In addition to the general goals described above, more specific design goals were related to the implementation of the SAS System:

- The core of the system should be compact and reentrant so that it can be installed in a read-only shareable system area. (IBM version)
- A minimum of operating system control language should be required.
- SAS should be able to group many data sets in a library-like file. In this way, data are managed more simply, and fewer operating system limitations are encountered. (IBM version)
- The SAS supervisor should provide the facilities and supporting interfaces so that the statistical and reporting procedures can be written in higher-level languages so they can be better written and maintained.
- Procedures should be separate modules so that there are no limits to their number and so they can be developed and maintained as individual units.

SYSTEM OVERVIEW

The SAS System is a library of modules tied together by a central supervisory program. The supervisor and its service routines provide a common way of handling basic system functions.

The supervisor brings in other modules as needed from the program library. There are two classes of modules that can be called in, depending on whether SAS is running a DATA step or a PROC step.

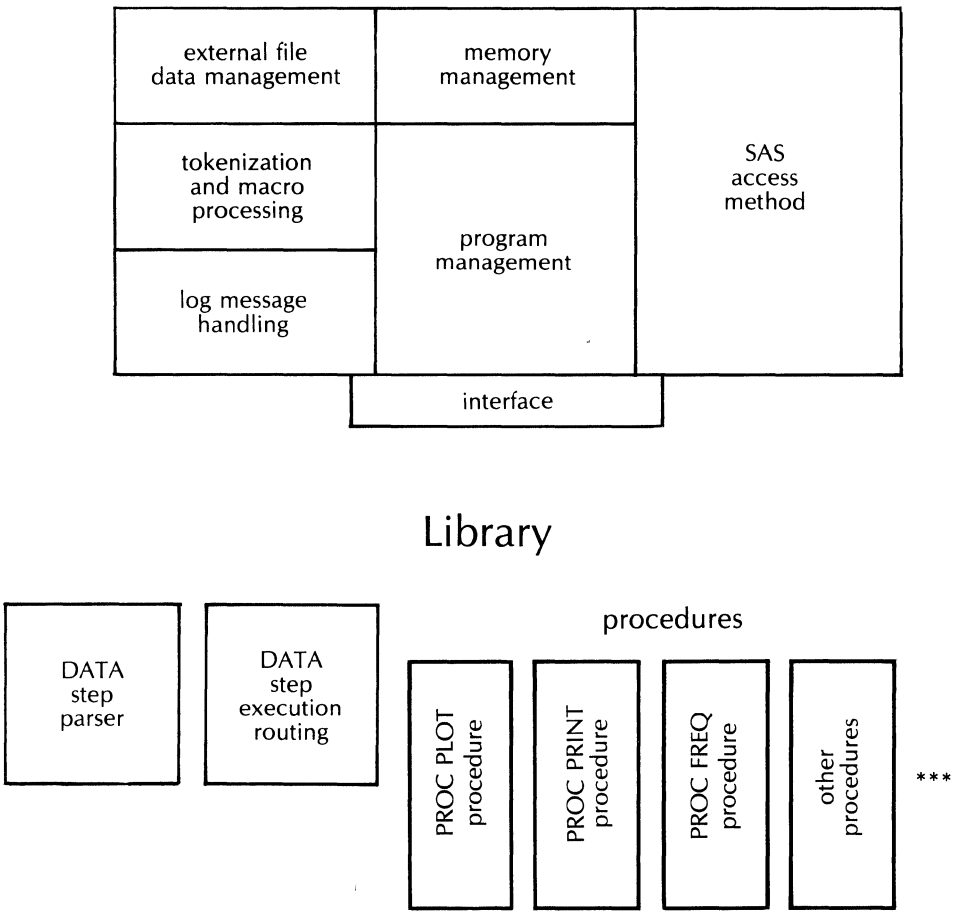


Figure 11.1 SAS Supervisor and Library

Implementation for the IBM 370

The supervisor and DATA step are written exclusively in reentrant System/360-370 assembler language. This provides SAS with the opportunity to be compact and efficient, especially for data-processing jobs. Procedures are written in PL/I.

The Supervisor

The supervisor handles

- program management, that is, initiating and terminating steps
- memory management
- access method services for SAS data libraries
- data management services for external files
- control language and macro processing
- log message handling
- global statements such as OPTIONS.

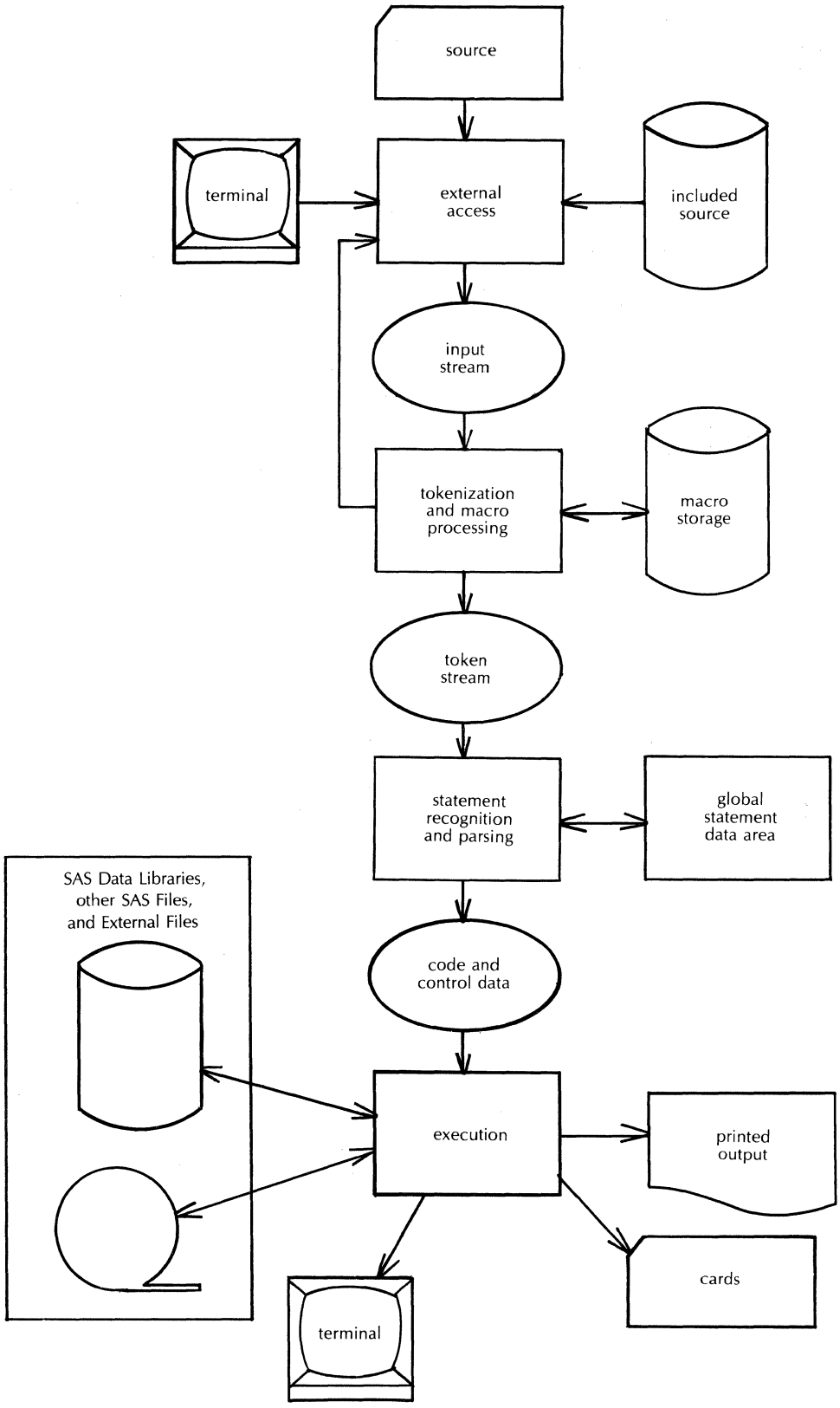


Figure 11.2 Control Flow

Execution Control Flow

The flow of data for the execution of a SAS job is

1. acquiring records (normal input, plus %INCLUDE data)
2. word scanning and macro expansion, also called *tokenization* or *lexical analysis*
3. statement recognition (by the supervisor or appropriate parser)
4. parsing the statement, producing code or control information
5. assembling the code fragments or control information.
6. execution.

SAS Data Library Access Method for the IBM 370

Under OS and VSE, SAS uses channel program-level access to manage data on direct-access storage devices in the form of SAS data libraries. Although SAS data libraries are marked with the attribute DSORG=DA, SAS does not use the BDAM access method (basic direct access method). (Preformatting a BDAM data set is too inefficient and, once formatted, it cannot be extended.)

A SAS data library starts with a directory file, which may have additional pieces (subextents) scattered later in the file. The directory records the locations and attributes of the SAS data sets and other logical files in the data library. Each logical file described by the directory is mapped into one or more physical "subextent" pieces in the OS or VSE data set.

A SAS data set is written to a SAS logical file in two pieces: the descriptor records describe the file and its variables, then the data records follow. A SAS data set can be extended to any length by acquiring additional "subextents" in the physical data set or blocks in the CMS file system. Empty space may appear in OS and VSE data libraries as files are deleted or replaced. SAS always uses the lowest unreserved space available to form new "subextents" to store data. In this way, the data set does not need to be "compressed" periodically. (See **Figure 11.3.**)

SAS FEATURES FOR BOTH DATA AND PROC STEPS

The remainder of this section of the manual describes SAS features that are applicable to both DATA and PROC steps. Topics discussed include SAS files and data sets, SAS informats and formats, missing values, the macro language, the SAS log, and output from procedures, and SAS Display Manager for full-screen terminals.

The following statements can be used anywhere in a SAS program, either outside of or within either DATA or PROC steps. The statements are divided into four groups: information and system control, listing/printout control, external services, and source language processing. The statements described in the chapter "SAS Macro Language" can also be used anywhere.

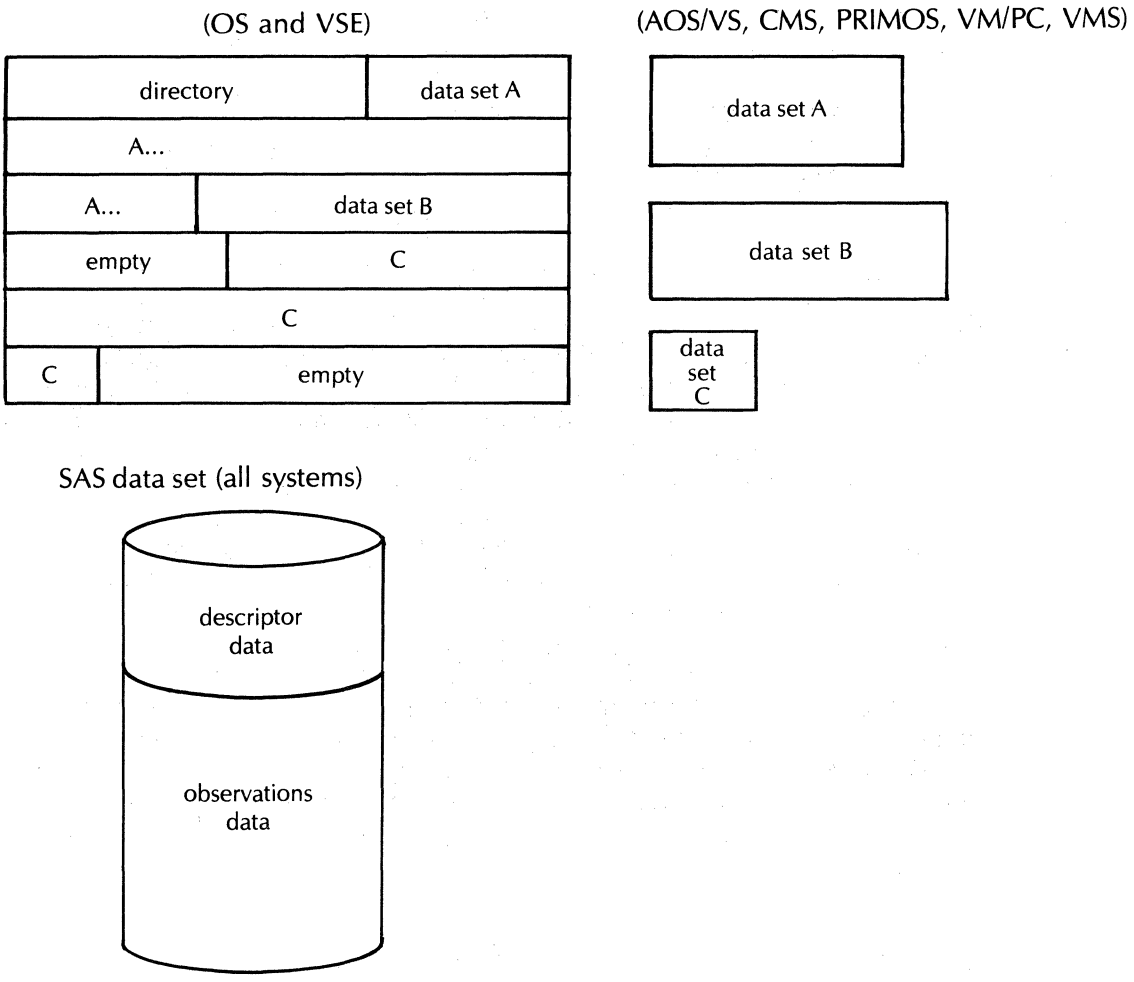


Figure 11.3 SAS Data Library

Information and system control

FILENAME defines an external file to be referenced in a FILE, INFILE, or %INCLUDE statement.

Operating systems: AOS/VS, PRIMOS, and VMS

FOOTNOTE specifies footnote lines to be written with SAS output.

Operating systems: All

HELP allows you to access additional information about SAS and your SAS installation.

Operating systems: All

LIBNAME defines a libname for a SAS data library to be used as the first level of a two-word SAS data set name and by procedures that access SAS files.

Operating systems: AOS/VS, PRIMOS, and VMS

LIBSEARCH defines a list of libnames that indicate the order in which SAS data libraries are to be searched for input SAS files.

Operating systems: AOS/VS, PRIMOS, and VMS

OPTIONS allows you to change SAS system options.

Operating systems: All

RUN causes the previously entered SAS step to begin execution.

Operating systems: All

TITLE specifies title lines to be written with SAS output.

Operating systems: All

Listing/printout control

CLEAR used interactively, causes the terminal screen to be cleared and execution to proceed.

Operating systems: CMS, OS, VM/PC, and VSE

comment allows you to write comments with your SAS statements to document the program.

Operating systems: All

PAGE skips to the top of the next page of the SAS log.

Operating systems: All

%PUT prints text on the SAS log.

Operating systems: All

SKIP skips a number of lines on the SAS log.

Operating systems: All

External services

CMS causes CMS commands to be executed during a SAS session.

Operating systems: CMS and VM/PC

TSO causes TSO commands or CLISTs to be executed during a SAS session.

Operating system: OS

X is used to issue a command from the operating system's control language within a SAS job.

Operating systems: AOS, CMS, OS, PRIMOS, VM/PC, and VMS

Source language processing

ENDSAS causes SAS to terminate execution immediately.

Operating systems: All

%INCLUDE includes SAS source lines from external or internal sources or entered from the terminal in a SAS program.

Operating systems: All

%LIST lists previously entered SAS statements.

Operating systems: All

MACRO allows you to name and store a segment of a SAS program and call it later with a macro name.

Operating systems: CMS, OS, VM/PC, and VSE

%RUN ends source statements included by a %INCLUDE statement.

Operating systems: All

Chapter 12

SAS® Statements Used Anywhere

CLEAR Statement

Operating systems: CMS, OS, VM/PC, and VSE

The CLEAR statement causes the terminal screen to be cleared and SAS execution to proceed. Use the CLEAR statement only in interactive (not display manager) sessions; otherwise, it is ignored.

The form of the CLEAR statement is

CLEAR [PAUSE] [SMON | SMOFF];

where

PAUSE tells the SAS System to wait for you to press ENTER on your terminal before proceeding. The screen is cleared **after** you respond so that you can review SAS output up to that point.

Operating systems: CMS, OS, VM/PC, and VSE

SMON | SMOFF controls whether the IBM product Session Manager is active, if that product is available to your terminal. If you begin an interactive SAS session under TSO using the Session Manager product, you can temporarily deactivate Session Manager with the command

```
CLEAR SMOFF [PAUSE];
```

Subsequent CLEAR statements clear the screen without causing Session Manager to be reinvoked. To reinvoke Session Manager, specify

```
CLEAR SMON [PAUSE];
```

You cannot use the SMON option to invoke Session Manager if you did not begin the SAS session under Session Manager.

You can also use the SMON | SMOFF options in an interactive SAS session that you began under the IBM product ISPF using Session Manager mode; however, you cannot enter Session Manager mode if you did not begin the SAS session under ISPF in Session Manager mode.

Operating system: OS

Unless you are using an IBM 3270-series or compatible terminal, the system option DEVICE= must be set to the type of terminal that you are using. (See "SAS Options" for a description of the DEVICE= option.) The following terminal device names are supported by the CLEAR statement (the names follow SAS/GRAPH conventions):

HP2647	TEK4012	TEK4024	TEK4052
HP2648	TEK4013	TEK4025	TEK4053
RAM6200	TEK4014	TEK4027	TEK4054
TEK4010	TEK4016		

Additional terminals may be supported at your installation. Check with your installation's SAS consultant to see if the CLEAR statement is available for your terminal.

CMS Statement

Operating systems: CMS and VM/PC

The CMS statement is used to invoke a CMS or CP command during a SAS job. The statement can appear in a DATA or PROC step or between steps in the job. The form of the CMS statement is

CMS [*CMScommand*];

or

CMS [**CP** *CPcommand*];

where

CMScommand is any CMS command.

CPcommand is any CP command.

The command is invoked immediately when the CMS statement is executed. Any CMS or CP command can be issued from the CMS statement as long as the statement does not destroy your environment. It is **not** necessary to precede EXEC names with the word EXEC.

If you omit *CMScommand* or CP *CPcommand*, as in the statement

```
CMS;
```

the SAS System enters the CMS subset. The system prompts you for CMS commands. When you want to return to the SAS session, issue the CMS command

```
RETURN
```

The CMS statement is treated as a comment when it appears in a job running under some other operating system.

See the *SAS Companion for the VM/CMS Operating System* for more information.

Comment Statement

Operating systems: All

The comment statement can be used anywhere in your SAS job to document the purpose of the job, to explain any unusual segments of the program, or to describe the steps in a complex program or calculation. The form of the comment statement is[1]

**message;*

where

message explains or documents the job. The message can be any length, although it cannot contain semicolons.

You can use any number of comment statements in a job. The comment statement **must** end in a semicolon. Macro variable references and macro calls are not expanded in SAS comments.

These statements are valid comments:

```
*THIS CODE FINDS THE NUMBER IN THE BY GROUP;
*TEST RUN;
*PROC PRINT;
    *VAR INSPECTR LOT;
```

Drawing a box around comments is a useful way to document your SAS job:

```
*------------------------------------------------------------*
|                    This is a box-style                     |
|                          comment                           |
*------------------------------------------------------------* ;
```

Comments of the form

*/*message*/*

can also be used. They can appear within SAS statements anywhere a single blank can appear and can contain semicolons. Do not nest comments of this type, and do not begin them in column 1. (The SAS System may interpret the symbols /* in columns 1 and 2 as a request to end the SAS job or session. See the **ENDSAS Statement** for more information.) Comments within a TITLE or FOOTNOTE statement are printed with the title.[2]

For example, the statement

```
PROC SORT /* SORT THE DATA SET */;
```

is a valid SAS statement.

You can also use comments to eliminate parts of an existing SAS program. The following example eliminates some BY variables for the current execution of the job:

```
PROC SORT;
   BY REGION STATE /* COUNTY CITY */;
```

This example eliminates a DATA and a PROC step:

```
DATA ALL;
   SET IN.OLD;
PROC PRINT;
```

```
        /*
DATA SUBSET;
   SET ALL;
   IF STATE='NC';
PRO PRINT;
   VAR CROP FARMSIZE;
        */
RUN;
```

To run the complete program, remove the comment symbols.

Notes

1. **CMS, OS, VM/PC, and VSE**: You can also write the comment statement as

```
COMMENT message;
```

The following comment statement shows this form:

```
COMMENT PRINT MAXIMUM EARNED THIS YEAR;
```

2. **CMS, OS, VM/PC, and VSE**: Comments within a title or footnote not surrounded by quotes are replaced with blanks when the title is printed.

ENDSAS Statement

Operating systems: All

The ENDSAS statement causes a SAS job or session to terminate at the end of the current DATA or PROC step.

The form of the ENDSAS statement is

ENDSAS;

You can also end a SAS job or session by placing the symbols

/*

in columns 1 and 2 of a blank line.

FILENAME Statement

Operating systems: AOS/VS, PRIMOS, and VMS

The FILENAME statement associates a SAS fileref (a file reference name) with an external file's complete name (directory plus file name).[1] The fileref is then used as a shorthand reference to the file in the SAS programming statements that access external files (INFILE, FILE, and %INCLUDE). Associating a fileref with an external file is also called *defining* the file.[2] Use the FILENAME statement to define a file before using a fileref in an INFILE, FILE, or %INCLUDE statement.[3] You must define each external file accessed; therefore, if you are accessing more than one external file, you must include more than one FILENAME statement.

The association between a fileref lasts only for the duration of the SAS job or session or until you change it by specifying another FILENAME statement. You can change the fileref for a file as often as you want.

The form of the FILENAME statement is

FILENAME *fileref 'filename'...* **;**

where

fileref specifies the fileref to be associated with the external file. The fileref must follow the rules for SAS names. A fileref can refer to only one file. If you are accessing more than one file from the same directory, each file must be assigned a unique fileref.

filename specifies the name of the external file. *Filename* must be a complete pathname that specifies the file name, not just the directory name.

The following program reads data lines from a file identified with the fileref FOOD and creates a temporary SAS data set:

```
FILENAME FOOD 'externalfile';
DATA MARKET;
   INFILE FOOD;
   INPUT DAY MMDDYY8. MEAT GROC DAIRY;
RUN;
```

The difference between the FILENAME statement and the LIBNAME statement is that the FILENAME statement defines a fileref for an external file to be used in a SAS FILE, INFILE, or %INCLUDE statement, whereas the LIBNAME statement defines a libref that your program uses to read or create permanent SAS files, such as SAS data sets and formats.

This example reads data from a file with fileref GREEN and creates a permanent SAS data set stored in the directory with libref SAVE:

```
FILENAME GREEN 'externalfile';
LIBNAME SAVE 'directory';
DATA SAVE.VEGETABL;
   INFILE GREEN;
   INPUT LETTUCE CABBAGE BROCCOLI;
RUN;
```

Notes

1. The FILENAME statement is also used to assign a libref to a transport library. See the "SAS Files" chapter.

2. **AOS/VS:** A file can also be defined by the host CREATE/LINK command. If a file is defined by both the CREATE/LINK command and the FILENAME statement, the information in the FILENAME statement takes precedence. That is, if the same file is given two different filerefs, the fileref specified by the FILENAME statement is used.

VMS: A file can also be defined by the host ASSIGN command. If a file is defined by both the ASSIGN command and the FILENAME statement, the information in the FILENAME statement takes precedence. That is, if the same file is given two different filerefs, the fileref specified by the FILENAME statement is used.

3. If a fileref is used in a FILE, INFILE, or %INCLUDE statement without having been defined, SAS uses the word specified as the fileref for a file name in your current default directory, as follows:

Operating System	FILE	INFILE	%INCLUDE
AOS/VS	fileref	fileref	fileref.SAS
PRIMOS	fileref	fileref	fileref.SAS
VMS	fileref.DAT	fileref.DAT	fileref.SAS

For example, if you use this INFILE statement:

```
INFILE STUDENT;
```

without first defining STUDENT as a fileref, SAS looks for a file called STUDENT (or STUDENT.DAT) in your current directory.

FOOTNOTE Statement

Operating systems: All

Use the FOOTNOTE statement to print lines at the bottom of the page. FOOTNOTE statements can be used to produce up to ten lines.

The FOOTNOTE statement has the form:

FOOTNOTE[*n*] ['*text*'];

where

n specifies the relative line to be occupied by the footnote. For footnotes, lines are "pushed up" from the bottom. The FOOTNOTE statement with the highest number appears on the bottom line. *N* can range from 1 to 10; if you omit *n*, SAS assumes a value of 1.

'text' specifies the text of the footnote.

Once you use a FOOTNOTE statement, the text is printed on all pages until the SAS System encounters another FOOTNOTE statement for that line.[1] When a FOOTNOTE statement is specified for a given line, it cancels the previous FOOTNOTE statement for that line and for all lines below it. To cancel all existing footnotes, specify

```
FOOTNOTE;
```

You can insert SAS/GRAPH commands into FOOTNOTE statements. If the FOOTNOTE statement is used with a non-graphics procedure, the commands are ignored. For example:

```
FOOTNOTE C=RED H=2 F=COMPLEX 'THIS IS A FOOTNOTE';
```

If this statement is executed with a procedure other than a graphics procedure, the SAS/GRAPH commands are ignored and only the quoted string appears in the footnote. If the statement is used with a graphics procedure that produces a graphic display, the commands are interpreted, and the footnote is produced accordingly.

Note

1. **CMS**, **OS**, **VM/PC**, and **VSE**: Footnotes are not printed by DATA steps that write to print files, including the standard SAS print file, or by procedures in the SUGI Supplemental Library.

HELP Statement

Operating systems: All

If the SAS HELP facility was selected by your installation at the time the SAS System was installed, then you can use the HELP statement to obtain additional information about SAS procedures, statements, and other features as well as the status of SAS products at your computing installation.

The form of the HELP statement is

HELP [*keyword*] [*/ option*];

where

keyword specifies the name of a SAS procedure, statement, or other feature about which you want to obtain additional information. For a complete list of possible keywords, enter the statement

```
HELP;
```

These special keywords can also be specified:

DSOPTION provides information about SAS data set options.

FONTS provides information about SAS/GRAPH fonts.

FORMATS lists SAS formats and their characteristics.

FUNCTION lists SAS functions and call routines and gives a brief description of each.

GOPTIONS provides information about SAS/GRAPH system options.

GRPNEWS gives the status of the current release of the SAS/GRAPH product.

INFORMAT lists SAS informats and their characteristics.

MACRO provides information about the SAS macro facility.

NEWS gives news supplied and maintained by your installation personnel.

OPERATOR gives the precedence of SAS DATA step operators.

OPTIONS gives information about SAS system options.

PROCS lists the SAS procedures for which HELP is available.

SASNEWS gives the status of the current SAS release at your installation. This information is provided initially for each major SAS release by SAS Institute and may be updated locally for each maintenance release.

SITEINFO provides information about your computing installation. This information is maintained by your installation personnel and normally includes the name of your installation's SAS consultant.

STMTS lists the SAS statements for which HELP is available.

VARLIST gives abbreviations for variable lists.

The option below can be specified in the HELP statement following a slash (/):

FS/NOFS temporarily overrides the FS/NOFS system option to specify whether HELP information is displayed in full-screen format on a display terminal. See "SAS Options" for information on the SAS system option FS/NOFS.[1]

For example, the statement

```
HELP VARLIST;
```

provides a brief description and the form of variable lists available in the SAS System. The statement

```
HELP SORT;
```

provides a brief description and the form of statements used in the SORT procedure.

Note

1. **OS**: If NOFS, NOFSP and NODMS are all specified at invocation, the FS option on the HELP statement is ignored.

%INCLUDE Statement

Operating systems: All

Introduction

The %INCLUDE (or %INC) statement is used to include SAS statements and data lines from external files, from earlier points in the same job or session, or from the terminal.

A %INCLUDE statement must begin at a statement boundary. That is, it must be the first statement in a SAS job or immediately follow a semicolon ending another statement. A %INCLUDE statement cannot immediately follow a CARDS or PARMCARDS statement; however, you can include data lines or parmcard lines with the technique described in **Including Data Lines or Parmcards** later in this section. A %INCLUDE statement can be a statement in a file that is %INCLUDEd. There is no limit to the level of nesting of %INCLUDE statements.

The difference between the %INCLUDE statement and the INCLUDE command in the display manager program editor is that the %INCLUDE statement sends the included statements to the SAS System for immediate execution, while the INCLUDE command brings the included lines into the program editor screen. You must issue a display manager SUBMIT command to send those lines to the SAS System for execution.

The form of the %INCLUDE statement is

%INCLUDE *source ... / options;*

or

%INC *source ... / options;*

where

source specifies the source from which the included statements are to come. *Source* can be one of three types: external, internal, and the terminal. You can specify any number of sources in a %INCLUDE statement, and you can mix the types of included sources. The types of sources are described later in this section.

options specifies one or more options that can be used when you include an external source. Some of the %INCLUDE statement options have the same function as SAS system options. However, the %INCLUDE statement options are effective only for the execution or duration of the %INCLUDE statement. In addition, an OPTIONS statement in the included source that contains a SAS system option overrides that option in the %INCLUDE statement. Individual options are described later in this section.

The types of sources that can be specified are

external includes input from a file stored on disk or tape. The SAS option INCLUDE must be in effect in order for external sources to be included. You must include an entire external file; you cannot selectively include lines. Once you have included an external source, the SAS statements and data lines in it receive line numbers. You can then include some or all of those lines as an internal source (see below). If the SOURCE2 option is in effect, the SAS statements from the external source are displayed on the SAS log. The SAS log also displays the fileref of the source and the level of nesting (1, 2, 3, and so on).

Including external sources is useful in all types of SAS processing: batch, display manager, interactive, and noninteractive.

The requirements for using an external source vary according to your operating system. See **Usage Notes for External Files** later in this section for an explanation of how to include an external source under your operating system.

internal includes lines entered earlier in the same SAS job or session.[1]

Including internal lines is most useful in display manager and interactive processing. Specify internal lines as:

n includes line *n*.

n-m *n:m* includes lines *n* through *m*.

For example, the statement

```
%INCLUDE 1 5 9-12 5 13-16;
```

causes SAS to process lines 1, 5, 9-12, 5 again, and 13-16 as though you had entered them once more at the terminal. Use a %LIST statement to determine the line numbers.[2]

terminal causes SAS to pause during display manager, noninteractive, or interactive processing and prompt you to enter statements from the terminal. Specify the terminal as:

*

as in

```
%INCLUDE *;
```

While SAS is expecting input from the terminal after a %INCLUDE * statement in an interactive session, it prompts you with a line number followed by an asterisk (*). (The asterisk does not appear in display manager.) To cause SAS to resume reading lines from the original source, enter a %RUN statement (described later in this chapter). In most cases, you specify the terminal as a source when you are preparing a file of SAS statements to be used in a later interactive, noninteractive, or display manager session.[3]

These options can appear in the %INCLUDE statement:

SOURCE2/ NOSOURCE2 — controls whether the SAS statements from external files in the current %INCLUDE statement appear on the SAS log, regardless of how the SAS system option SOURCE2/NOSOURCE2 is specified. (See "SAS Options" for more information about the SOURCE2/ NOSOURCE2 system option.)

Operating systems: CMS, OS, VM/PC, and VSE.

JCLEXCL — ignores any lines of control language in the included source.[4]

Operating systems: OS and VSE.

S2=*length* — specifies the length of the record to be used for input. (See "SAS Options" for more information about the S2= system option.)

Operating systems: CMS, OS, VM/PC, and VSE.

Including an external file For example, suppose you regularly create and print a SAS data set containing variables X, Y, Z, and MONTH. You can store the SAS statements for the DATA and PROC steps in a file for later execution. The file contains:

```
DATA MONTHLY;
   INFILE NEW;
   INPUT X Y Z MONTH$;
PROC PRINT;
RUN;
```

To execute this group of statements in your current job or session, assign a fileref (for example, IN1) to the file as described in **Usage Notes for External Files**. Then use that name in the %INCLUDE statement:

```
%INCLUDE IN1;
```

Including input from the terminal If you want to be able to insert more statements into a SAS program each time you include it, store a %INCLUDE * statement as one of the statements in the file, for example:

```
DATA MONTHLY;
   INFILE NEW;
   INPUT X Y Z MONTH$;
```

```
   %INCLUDE *;
PROC PRINT;
RUN;
```

When you execute this file of SAS statements, SAS prompts you to enter statements at the terminal. You can then enter statements such as:

```
IF MONTH='JANUARY';
TITLE 'DATA FOR MONTH OF JANUARY';
%RUN;
```

The %RUN statement causes SAS to resume processing statements in the file.

Including data lines or parmcards To include data lines or parmcards, place the CARDS or PARMCARDS statement in a file and the data lines or parmcard lines in another file. Then use both sources in a single %INCLUDE statement. For example, suppose a file described with fileref F1 contains the SAS statement

```
CARDS;
```

and a file described with fileref F2 contains the data lines you want to process in the current DATA step. Then these statements let you include the data lines:

```
DATA ONE;
   INPUT X Y Z;
   %INCLUDE F1 F2;
RUN;
```

You can also make the CARDS or PARMCARDS statement the first line in the file containing the data or include a CARDS or PARMCARDS statement saved earlier in the current session.

Variable in a TITLE or FOOTNOTE statement You can include the value of a variable in a SAS TITLE or FOOTNOTE statement using the %INCLUDE statement. The statements below first create a variable named NOW containing the current date and LASTWK containing the date a week ago. Then the PUT statements write a TITLE and a FOOTNOTE statement to an external file with fileref NEW. (You must define the file according to the rules in **Usage Notes for External Files.**) In the PROC step, the %INCLUDE statement reads the file containing the TITLE and FOOTNOTE statements.

```
DATA _NULL_;
   NOW=TODAY();
   LASTWK=NOW-7;
   FILE NEW;
   PUT 'TITLE ''DAILY REPORT FOR ' NOW DATE. ''';';
   PUT 'FOOTNOTE ''SUMMARIZING ACTIVITY SINCE ' LASTWK DATE. ''';';
RUN;
PROC PRINT DATA=ANYDATA;
   %INCLUDE NEW;
RUN;
```

Note: you can simplify the PUT statements when the DQUOTE option is in effect, as follows:

```
*OPTIONAL METHOD WHEN DQUOTE IS IN EFFECT;
PUT 'TITLE "DAILY REPORT FOR ' NOW DATE. '";';
PUT 'FOOTNOTE "SUMMARIZING ACTIVITY SINCE ' LASTWK DATE. '";';
```

In this method the PUT statements use the single quotes and the TITLE and FOOTNOTE statements in file NEW contain the double quotes (quotation marks).

You can use the method shown in this example to write SAS programs that generate and execute other SAS programs.

You can also use macro variables to include a variable in a TITLE or FOOTNOTE statement. See "SAS Macro Language" for details.

Usage Notes for External Files

AOS/VS Specify an external source as

fileref

where SAS associates *fileref* with a file according to the following rules:

- If you associated *fileref* with an external file in a previous FILENAME statement, the %INCLUDE statement reads from that external file. For example, the statements

```
FILENAME SPORTS ':UDD:userdir:BASEBALL.SAS';
%INCLUDE SPORTS;
```

  read lines from :UDD:*userdir*:BASEBALL.SAS.
- If you have not associated *fileref* with an external file, SAS associates *fileref* with a file named *fileref*.SAS in your current default directory. For example, the statement

```
%INCLUDE SPORTS;
```

  reads from the file SPORTS.SAS in your current default directory.

CMS, VM/PC Specify an external source as

filename

or

fileref

or

fileref(*member*)

where

filename
: is a CMS filename. If the file identified by this filename has a filetype of SAS and the filename does not duplicate the fileref/filetype of a SAS data set, SAS automatically issues a FILEDEF for the file. This is true regardless of the file's filemode. For any file that does **not** have filetype SAS, you must issue a FILEDEF command before the %INCLUDE statement.

  For example, if you want to %INCLUDE a file of source statements with file-id MEANS SAS B, use the following %INCLUDE statement:

```
%INCLUDE MEANS;
```

  Since the filetype is SAS, you do not need to issue a FILEDEF as long as you have not used MEANS as a fileref for a SAS data set.

fileref
: identifies the fileref of a file containing SAS programming statements. You must issue a FILEDEF command for the file before executing the %INCLUDE statement. For example, to use a file called SOURCE STUDY A, issue a FILEDEF first:

```
FILEDEF STMTS DISK SOURCE STUDY A
```

Then use this %INCLUDE statement:

```
%INCLUDE STMTS;
```

fileref(member)
identifies the fileref and one or more members of a partitioned data set (a MACLIB, TXTLIB, or OS data set). You must issue a FILEDEF for the data set before executing the %INCLUDE statement. Suppose you want to use a MACLIB member called CODE. First issue a FILEDEF command:

```
FILEDEF PROGRAM DISK MYSAS MACLIB A
```

Then use this %INCLUDE statement:

```
%INCLUDE PROGRAM(CODE);
```

OS Specify an external source as

fileref

or

fileref(member, ...)

where

fileref
identifies either a sequential file or a member of a partitioned data set. For example,

```
%INCLUDE A1;
%INCLUDE IN MONDAY TUESDAY;
```

You must specify each fileref as the DDname parameter in a JCL DD statement or TSO ALLOCATE command in order to use the fileref in a %INCLUDE statement. If the fileref refers to a member of a partitioned data set, specify the member name as part of the data set name parameter in the JCL DD statement or TSO ALLOCATE command.

fileref(member, ...)
identifies one or more members of a partitioned data set. For example,

```
%INCLUDE IN(STUDY);
%INCLUDE SURVEY1(JAN,FEB,MAR);
```

You must specify each fileref as the DDname parameter in a JCL DD statement or TSO ALLOCATE command in order to use the fileref in a %INCLUDE statement.

PRIMOS Specify an external source as

fileref

where SAS associates *fileref* with a file according to the following rules:

- If you associated *fileref* with an external file in a previous FILENAME statement, the %INCLUDE statement reads from that external file. For example, the statements

```
FILENAME SPORTS 'directory>BASEBALL.SAS';
```

```
%INCLUDE SPORTS;
```

read lines from *directory*>BASEBALL.SAS.

- If you have not associated *fileref* with an external file, SAS associates *fileref* with a file named *fileref*.SAS in your current default directory. For example, the statement

```
%INCLUDE SPORTS;
```

reads from the file SPORTS.SAS in your current default directory.

VMS Specify an external source as

fileref

where SAS associates *fileref* with a file according to the following rules:

- If you associated *fileref* with an external file in a previous FILENAME statement, the %INCLUDE statement reads from that external file. For example, the statements

```
FILENAME SPORTS '[directory]BASEBALL.SAS';
%INCLUDE SPORTS;
```

read lines from [*directory*]BASEBALL.SAS.

- If a FILENAME statement is not present but you associated *fileref* with an external file in a host operating system command, the %INCLUDE statement reads from that external file. For example, the statements

```
X 'ASSIGN [directory]BASEBALL.SAS SPORTS';
%INCLUDE SPORTS;
```

read lines from [*directory*]BASEBALL.SAS.

- If you have not associated *fileref* with an external file, SAS associates *fileref* with a file named *fileref*.SAS in your current default directory. For example, the statement

```
%INCLUDE SPORTS;
```

reads from the file SPORTS.SAS in your current default directory.

VSE Specify an external source as

fileref

or

sublibrary(*member*, ...)

where

fileref

includes statements from an external sequential file described in the job control by *fileref*. The record length of the sequential file must be 80 bytes. The block size of the sequential file must be the same as the current value of the SAS system option FILEBLKSIZE(2314). (This is also the default blocksize used to write sequential disk files using FILE and PUT statements.) You can use your installation's default for this parameter or you can specify it in a SAS OPTIONS statement or as a parameter of the EXEC statement in the control language for an individual job, for example:

```
// EXEC SASVSE,SIZE=nnK,PARM='FILEBLKSIZE(2314)=nnnn'.
```

sublibrary(*member, ...*)
includes statements from one or more source statement library members.

The source statement libraries that are searched for members are those specified in the SEARCH parameter of a LIBDEF SL JCL statement. If you are running under ICCF, the LIBDEF statement must be provided in the ICCF start-up JCL because the LIBDEF function is not provided by ICCF job entry statements.

This example includes SAS statements from a sequential file:

```
// DLBL IINC031,'datasetname'
// EXTENT SYS031,volumeserial
// ASSGN SYS031,DISK,VOL=volser,SHR
// EXEC SASVSE,SIZE=nnK,PARM='FILEBLKSIZE(2314)=80'
%INCLUDE IINC031/SOURCE2;
/*
```

This example includes SAS statements from source statement library members A.INCLLS and S.PROG01:

```
// DLBL TSTSSL,'datasetname'
// EXTENT ,volser
// DLBL PRODSSL, 'datasetname'
// EXTENT ,volser
// LIBDEF SL,SEARCH=(TSTSSL,PRODSSL)
// EXEC SASVSE,SIZE=nnK
SAS statements
%INCLUDE A(INCLLS);
%INCLUDE S(PROG01);
more SAS statements
/*
```

Notes

1. **AOS/VS, PRIMOS, and VMS**: All lines in a SAS session are saved automatically and are available for use with the %INCLUDE statement.

CMS, OS, VM/PC, and VSE: The SAS option SPOOL must be in effect to cause lines to be saved. However, the SPOOL option does not have to be in effect when you use the %INCLUDE statement. See "SAS System Options" for a description of the SPOOL option.

2. **AOS/VS, PRIMOS, and VMS**: SAS makes a new copy of lines included with a %INCLUDE statement. To use the same lines in a later %INCLUDE statement, give either the numbers of the original lines or the numbers of the copies.

CMS, OS, VM/PC, and VSE: To include some of the same lines in a later %INCLUDE statement, use the original line numbers. Using lines in a %INCLUDE statement does not assign them new line numbers.

3. **AOS/VS, CMS, PRIMOS, VM/PC, and VMS**: You cannot use a %INCLUDE* statement in a batch job.

OS and VSE: You can use a %INCLUDE * statement in a batch job by creating a file with the fileref SASTERM that contains the statements you would otherwise enter from the terminal. The %INCLUDE * statement causes SAS to read from the SASTERM file. Insert a %RUN statement into the SASTERM file where you want SAS to resume reading from the original source. Subsequent %INCLUDE* statements resume reading from the SASTERM file with the statement following the most recent %RUN statement. The last statement in the SASTERM file must be a %RUN statement.

4. **OS**: Control language contains two slashes (//) or a slash and asterisk (/*) in columns 1 and 2.

VSE: Control language contains two slashes (//), a slash and asterisk (/*), or a slash and ampersand (/&) in columns 1 and 2, or an asterisk, space, and two dollar signs (* $$) in columns 1 through 4.

LIBNAME Statement

Operating systems: AOS/VS, PRIMOS, and VMS

The LIBNAME statement defines one or more directories (or path names) to be used by your SAS program. If you want to read or create a permanent SAS file, you must define the directory to be used in a LIBNAME statement. The LIBNAME statement associates a *libref* (short for "library reference") with a directory name. Then, you specify the libref in SAS statements to refer to the directory. You must include the LIBNAME statement before a SAS statement that specifies the libref.

Refer to "SAS Files" for information on different types of SAS files and how they are logically grouped to form a SAS data library within a directory.

Use the LIBNAME statement to define directories that your program accesses to read or create SAS files, such as SAS data sets and permanently stored formats. If your program reads and creates external files (for example, in INFILE and FILE statements,) use the FILENAME statement to define the file. See the **FILENAME Statement** earlier in this chapter.

After you associate a libref with a directory in the LIBNAME statement, you can use the libref in a LIBSEARCH statement to place the associated directory in a search list. See the **LIBSEARCH Statement** later in this chapter.

The form of the LIBNAME statement is

LIBNAME *libref 'directory'* ... ;

where

libref is a name associated with a directory and then used in SAS statements to refer to the directory. The libref must follow the rules for SAS names (as described in "SAS Files"). A libref must refer to only one directory, but you can refer to a single directory with more than one libref. For example,

```
LIBNAME DAY1 'directory1';
LIBNAME DAY2 'directory1';
```

DAY1 and DAY2 refer to the same directory.

directory is the name of a directory to be accessed by the SAS program. The directory name must be enclosed in quotes and must include the appropriate character to separate elements in the directory name.[1] If you are defining a directory located on tape, use the tape device name as the directory name.

The association between the libref and the directory remains in effect until you change it with another LIBNAME statement or until the end of your session. You can either create or read a permanent SAS file once you have defined the directory in a LIBNAME statement. The following program creates a permanent SAS data set: `th,10;`

```
LIBNAME STORE1 'yourdirectory';
DATA STORE1.MAR;
   INPUT DAY MMDDYY8. MEAT GROC DAIRY;
   CARDS;
data lines
;
```

The LIBNAME statement associates the libref STORE1 with *yourdirectory*, and then STORE1 is used as the first level of the permanent SAS data set name in the DATA statement. When this program executes, the permanent SAS data set MAR. SSD is stored in *yourdirectory*.

You can define more than one directory in the same LIBNAME statement. The following DATA step reads a permanent SAS data set from one directory and creates a permanent SAS data set in another directory. One LIBNAME statement defines both directories:

```
LIBNAME STOCK 'directory1' FOOD 'directory2';
DATA STOCK.APPLES;
   SET FOOD.FRUIT;
   IF ITEM='GRANNY' | ITEM='TOSH' THEN OUTPUT;
```

Once the association between libref and directory is made, you can access any SAS file in the directory by using the libref as the first level of a permanent SAS file name. For example, suppose you want to append one SAS file to another SAS file in the same directory. First, use the LIBNAME statement to associate the libref with the appropriate directory. Then, invoke the APPEND procedure, as follows:

```
LIBNAME SALES 'userdirectory';
PROC APPEND BASE=SALES.YR1985 DATA=SALES.FEB;
```

This program appends the SAS file FEB to the SAS file YR1985. Since both files are located in the directory referenced by SALES, the first-level name of both permanent SAS files is the same.

To store user-defined formats, use the LIBNAME statement to define the directory before you invoke the FORMAT procedure. The following program creates a format named SECT. in *directory1*:

```
LIBNAME FMT 'directory1';
PROC FORMAT LIBRARY=FMT;
   VALUE SECT 1 = 'COASTAL'
              2 = 'PIEDMONT'
              3 = 'MOUNTAIN';
```

Notice that the libref FMT is associated with *directory1* in the LIBNAME statement and then used as the value of the DDNAME= option in the PROC FORMAT statement. If the DDNAME= option is omitted, SECT. is stored as a temporary format in the WORK library.

To use a permanent user-defined format, you must place the directory containing the format in a search list after you have defined the directory in a LIBNAME statement. Use the LIBSEARCH statement to place the directory in a search list. See the **LIBSEARCH Statement** and "The FORMAT Procedure" for more information on creating and using permanently stored user-defined formats.

Note

1. The character used to separate levels of the directory name is different for each operating system:

   AOS/VS: LIBNAME *libref ':UDD:userdir:dir2';*
   PRIMOS: LIBNAME *libref '<mfd>userdir>dir2';*
   VMS: LIBNAME *libref '[userdir.dir2]';*

LIBSEARCH Statement

Operating systems: AOS/VS, PRIMOS, and VMS

Use the LIBSEARCH statement to specify a list of librefs that refer to directories to be searched for SAS files, such as SAS data sets and user-defined formats, being used as input. The directories are searched in the order that you specify the librefs in the LIBSEARCH statement. The SAS System searches all directories referenced in the LIBSEARCH statement before searching the WORK library. (The WORK library is usually your current default directory unless you change the value of the WORK= option. See "SAS System Options" for more information on the WORK= option.)

Before using a libref in the LIBSEARCH statement, you must define it in a LIBNAME statement (described earlier in this chapter). If you use an undefined libref in a LIBSEARCH statement, the SAS System issues a warning message, ignores the undefined libref, and searches remaining directories. A libref that defines a SAS data library on tape cannot be listed in a LIBSEARCH statement.

The LIBSEARCH statement has the following form:

LIBSEARCH [*libref* ...]**;**

where

libref identifies a directory to be searched. The libref must be associated with the directory in a LIBNAME statement before it is specified in the LIBSEARCH statement. Librefs listed in a LIBSEARCH statement make up the *search list*. For example, the statement

```
LIBSEARCH GUAVA QUINCE KIWI;
```

defines a search list containing three directories.

A null LIBSEARCH statement, for example,

```
LIBSEARCH;
```

cancels the search list defined by the last LIBSEARCH statement. Only one search list can be in effect at a time. The search list specified in the last LIBSEARCH statement is the *current search list*, which replaces the previous search list. In the example

```
LIBSEARCH A;
LIBSEARCH B;
```

the directory identified by libref B is the only directory in the current search list. To include both directories, specify:

```
LIBSEARCH A B;
```

Suppose you want to read a SAS data set when a search list is in effect. First, create a search list with the following statements:

```
LIBNAME RED 'directory1' BLUE 'directory2' GREEN 'directory3';
LIBSEARCH RED BLUE GREEN;
```

Then, when a one-level SAS data set name is used in a SET, MERGE, or UPDATE statement, the SAS System searches *directory1*, *directory2*, and *directory3*, in that order, for the specified data set. If a permanent SAS data set with the specified

name exists in more than one directory, SAS uses the data set from the directory referenced earliest in the list. Therefore, the order of the librefs in the list may be important.

For example, if your program contains the statements

```
LIBSEARCH RED BLUE GREEN;
DATA NEW;
   SET OLD;
```

the SAS System searches for the permanent SAS data set OLD.SSD in *directory1*. If OLD.SSD is not found in *directory1*, the SAS System looks in *directory2*. If unsuccessful, it looks in *directory3* for the data set. If the permanent data set OLD.SSD does not exist in any of the directories in the search list, the SAS System looks for the temporary SAS data set OLD.SSW in the WORK library.[1]

To read a SAS data set located in a specific directory while a search list is in effect, use a two-level data set name. For example, if you specify

```
LIBSEARCH RED BLUE GREEN;
DATA NEW;
   SET GREEN.OLD;
```

the SAS System uses the permanent SAS data set OLD.SSD in the directory identified by libref GREEN, even if there is a data set named OLD.SSD in the directories referenced by RED and BLUE.

Note: the LIBSEARCH statement does not affect where newly created SAS data sets are stored. Therefore, the SAS data set NEW created in the example above is a temporary SAS data set stored in the WORK library.

To use a temporary SAS data set in the WORK library without searching directories in the current search list, specify the libref WORK as the first level of the two-level data set name. (WORK is the default libref for temporary SAS data sets.) For example,

```
LIBSEARCH ONE TWO THREE;
DATA TESTA;
   X=1;
RUN;
DATA TESTB;
   SET WORK.TESTA;
RUN;
```

In this program the SET statement reads the temporary SAS data set TESTA.SSW created by the previous DATA step. The directories identified by librefs ONE, TWO, or THREE are not searched for a permanent SAS data set named TESTA.SSD.

To access a permanently stored user-defined format, you must place the directory containing the format in a search list by using the LIBSEARCH statement. Suppose you have stored the format SECT. in the directory *directory1*. (See "The FORMAT Procedure" for information on how to store permanent and temporary user-defined formats.) To use format SECT. in a SAS program, you must first use the LIBNAME statement to associate a libref with *directory1* containing the format and then list the libref in a LIBSEARCH statement, as follows:

```
LIBNAME FMT 'directory1';
LIBSEARCH FMT;
PROC PRINT DATA=AREA;
   FORMAT GEO SECT.;
```

To access formats from more than one directory, include a libref for each directory in the LIBSEARCH statement. This example uses format ABC. from *directory1* and format XYZ. from *directory2*.

```
LIBNAME FMT 'directory1' FMT2 'directory2';
LIBSEARCH FMT FMT2;
PROC PRINT DATA=STUDY;
   FORMAT X ABC. Y XYZ.;
```

If two or more directories contain user-defined formats of the same name, the SAS System uses the format from the directory whose libref is listed earliest in the LIBSEARCH statement. If the user-defined format is not found in the directories in the search list, the SAS System assumes that the format is temporary and searches the WORK library.

Note

1. **AOS/VS, PRIMOS, and VMS**: SSD is an extension or suffix to a permanent SAS data set name, and SSW is an extension or suffix to a temporary SAS data set name. SAS uses these extensions (and others) to distinguish between different types of SAS files in a directory. For more information refer to the SAS documentation specifically for your operating system.

%LIST Statement

Operating systems: All

The %LIST statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to list lines entered earlier in the current job or session.[1]

The %LIST statement is most useful in interactive and display manager processing.

The form of the %LIST statement is

%LIST [*n*[:*m*]];

or

%LIST [*n*[-*m*]];

where

n lists line *n*.

n-m
n:m lists lines *n* through *m*. For example, the %LIST statement

```
%LIST 10-20;
```

lists lines 10 through 20.

Omitting a line value causes the SAS System to list all lines saved in the session.

Note

1. **AOS/VS, PRIMOS, and VMS**: All SAS statements and data lines are saved automatically and can be displayed with the %LIST statement.

CMS, OS, VM/PC, and VSE: The SAS option SPOOL must be in effect to cause lines to be saved. However, the SPOOL option does not have to be in effect when you use the %LIST statement. See "SAS System Options" for a description of the SPOOL option.

MACRO Statement

Operating systems: CMS, OS, VM/PC, and VSE

Introduction

The MACRO statement allows you to name and store a segment of a SAS program, then substitute that name for the program segment wherever the segment is to appear later in the job. The segment can be part of a SAS statement, a complete statement, or several statements. The stored segment is called a *macro*.

The MACRO statement, a feature of earlier versions of the SAS System, should not be confused with the %MACRO statement, used to define macros in the SAS macro language. Although this MACRO statement is still supported, the SAS macro language provides much more power and flexibility. While the MACRO statement allows only text substitution, the macro language allows text substitution, parameters, conditional execution of macros (including macro programming statements and macro functions), and user input during macro execution. See "SAS Macro Language" for more information.

The form of the MACRO statement is

MACRO *name text***%**

The MACRO statement ends with a % sign, not a semicolon. These terms may be used with a macro statement:

name names the macro. The name specified must be a valid SAS name.

text specifies the macro text, which may include part of a SAS statement, a complete SAS statement, or several SAS statements. To include a percent sign (%) as part of a macro without signaling the end of the macro, code two percent signs (%%):

```
MACRO PCT IF X=10 THEN A='EQUALS 10%%';%
```

There is no limit to the length of a macro, and a macro definition can appear anywhere in a SAS program. However, before a macro name can be used, it must be defined. If you create two macros with the same name in a single job, the second macro replaces the first.

Statements within a macro are not executed, nor is syntax checked, until the macro name appears in the program (the macro call). Data lines or lines following a PARMCARDS statement can not be included in a macro. There is no limit on the number of levels of macro nesting allowed.

Macro names enclosed in single or double quotes are not expanded, regardless of whether the DQUOTE option is in effect. Thus, if you enclose the text of TITLE and FOOTNOTE statements in quotes, you cannot use a macro to change the text.

Printing macro expansions The SAS System does not normally print the statements included in a macro on the SAS log when the macro name appears in the job. You can list expanded macros by specifying the system option MACROGEN. See "SAS Options" for a description of the MACROGEN option.

Macro libraries You can simulate a macro library facility using the %INCLUDE statement (described earlier in this chapter) or by putting your macros in a sequential data set that is concatenated to the front of the SYSIN file.[1]

Macro statements You can use the following statements for macros created using the MACRO statement. All of the statements are preceded by a % sign and are executed immediately.

%ACTIVATE Statement

Operating systems: CMS, OS, VM/PC, and VSE

The %ACTIVATE (or %ACT) statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to give the names of one or more macros for which you want recognition restored. Only macros defined by a MACRO statement can be specified.

The form of the %ACTIVATE statement is

%ACTIVATE *macronames*;

or

%ACT *macronames*;

where

macronames gives the names of one or more macros that you want recognized.

The %ACTIVATE statement is used to restore recognition of a macro whose name appears in a previous %DEACTIVATE statement.

%DEACTIVATE Statement

Operating systems: CMS, OS, VM/PC, and VSE

The %DEACTIVATE (or %DEACT) statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to give the names of one or more macros for which you want recognition temporarily suspended. Only macros defined by a MACRO statement can be specified.

The form of the %DEACTIVATE statement is

%DEACTIVATE *macronames*;

or

%DEACT *macronames*;

where

macronames gives the names of one or more macros that you want temporarily unrecognized. Deactivated macro names

still appear in the macro directory and, when listed with the %MLIST statement, are flagged with an asterisk. The macro can be reactivated by a %ACTIVATE statement.

%DELETE Statement

Operating systems: CMS, OS, VM/PC, and VSE

The %DELETE (or %DEL) statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to remove macros from the macro directory. Only macros defined by a MACRO statement can be specified.

The form of the %DELETE statement is

%DELETE *macronames*;

or

%DEL *macronames*;

where

macronames gives the names of one or more macros that you want deleted.

%MLIST Statement

Operating systems: CMS, OS, VM/PC, and VSE

The %MLIST (or %LISTM) statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to specify the name of a macro that you want listed or to list the macro directory.

The form of the %MLIST statement is

%MLIST *macroname*;

or

%LISTM *macroname*;

where

macroname gives the name of the macro you want listed. If no macro name is given, the names of all currently defined macros are listed.

Multiple Invocations of a Procedure: Example

Suppose you want to run the DISCRIM procedure several times with different class variables but the same PROC DISCRIM statement. You can define the macro DISCMAC:

```
MACRO DISCMAC
   PROC DISCRIM POOL=TEST LIST;
   CLASS CLASSMAC;
   VAR X Y Z; %
```

The CLASS statement calls a macro CLASSMAC, which can be redefined each time DISCMAC is invoked. If the class variables are AGE, SEX, and RACE, you can run DISCRIM for each variable using these statements:

```
MACRO CLASSMAC AGE %
DISCMAC
MACRO CLASSMAC SEX %
DISCMAC
MACRO CLASSMAC RACE %
DISCMAC
```

See "SAS Macro Language" for a description of macro variables in the SAS macro facility, which could be used in this application.

Note

1. **VSE:** VSE data sets cannot be concatenated. The macros must be included at the front of the input stream.

OPTIONS Statement

Operating systems: All

Introduction

The OPTIONS statement temporarily changes one or more system options from the default value set by your installation. (To obtain a list of the default values set by your installation use the OPTIONS procedure.)

The form of the OPTIONS statement is

OPTIONS *option...*;

where

option specifies one or more system options you want to change. The list of available system options is presented in "SAS System Options."

The changes made by an OPTIONS statement are in effect for the duration of the SAS job or until they are changed by another OPTIONS statement.

For example, suppose you want to suppress the date normally printed on SAS output pages and left align the output on the page. You can use this OPTIONS statement:

```
OPTIONS NODATE NOCENTER;
```

Some system options can only be changed when you invoke the SAS System by specifying options in your operating system control language or interactive SAS command. See "SAS System Options" for details.

An OPTIONS statement can be entered at any place in a SAS program. An OPTIONS statement entered within a DATA step or PROC step is effective for the execution of the entire DATA or PROC step, as well as for the duration of the SAS job, even if you enter the OPTIONS statement **after** all other statements (except the RUN statement) of the step. The following list gives a brief description of system options most frequently needed by SAS users. These options are described more fully in "SAS System Options." Note: if a negative form of the option is presented below the command, use this form to achieve the opposite effect of the use described.

Frequently Used Options

SAS input stream format options

CAPS
NOCAPS
: translates lowercase SAS input into uppercase.

CHARCODE
NOCHARCODE
: substitutes characters for terminals that lack underscore (_), vertical bar (|), or logical not sign (¬).

DQUOTE
NODQUOTE
: allows character literals bounded by either single quotes (') or double quotes (").

Options for data sets

FIRSTOBS=
: specifies first observation to be processed from SAS data sets.

GEN=
: specifies the number of generations of history records to save for each SAS data set.

OBS=
: specifies last observation to be processed from SAS data sets.

REPLACE
: replaces permanently allocated SAS data sets with others of the same name.

USER=
: specifies the first-level name assumed for all one-level SAS data set names in a SAS program.

Output formatting options

CENTER
: centers SAS procedure output.

DATE
NODATE
: prints the date at the top of the page.

INVALIDDATA=
: specifies the value that the SAS System assigns to a variable when invalid input data are encountered.

LABEL
NOLABEL
: specifies that variable labels are to be available to SAS procedures.

LINESIZE=
: gives printer line width for log and procedure output.

MISSING=
: specifies the character the SAS System prints for missing numeric variable values.

NUMBER
NONUMBER
: prints page numbers on SAS output.

OVP
NOOVP
: overprints SAS output.

PAGESIZE=
: gives number of lines printed per page of SAS output.

SKIP=
: specifies the number of lines to skip at the top of the page for printed SAS output.

TLINESIZE=
: gives the linesize for displaying SAS output on a terminal.

Other options

ERRORABEND
: determines whether SAS abends for errors that normally cause SAS to issue an error message.

ERRORS=
: specifies the maximum number of observations for which complete error messages are printed.

FMTERR
: controls whether SAS supplies a default format when it cannot find a format associated with a variable.

PAGES=
: gives the maximum pages printed as output from SAS jobs.

Source list options

SOURCE
: lists SAS source statements on the SAS log.

SOURCE2
: lists secondary source statements from files included by %INCLUDE on the SAS log.

Macro options

MACROGEN
: prints text generated by macros.

MPRINT
: controls whether statements produced by macro execution are printed.

SYMBOLGEN
: prints text generated by macro variable expansion.

PAGE Statement

Operating systems: All

The PAGE statement is used to skip to a new page on the log.[1]
The form of the PAGE statement is

PAGE;

The PAGE statement has no effect in display manager.

Note

1. **AOS/VS, PRIMOS, and VMS**: The PAGE statement is listed at the top of the new page on the SAS log.
CMS, OS, VM/PC, and VSE: The PAGE statement is not listed on the SAS log.

%PUT Statement

Operating systems: All

The %PUT statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to send a message to the SAS log or to the terminal screen.

The form of the %PUT statement is

%PUT *message*;

where

message is any message you want printed on the log.

Important note The SAS macro language also contains a %PUT statement. If the MACRO option is in effect, using a %PUT statement invokes the %PUT statement from the macro language, even if no other macro actions are present in the job or session. The %PUT statement described here is available **only** if the NOMACRO option (which makes the SAS macro language unavailable) is in effect.

The following table describes the differences between the SAS language %PUT statement and the SAS macro language %PUT statement.

Table 12.1 Comparison of the SAS %PUT Statement and the Macro Language %PUT Statement

SAS %PUT Statement	Macro Language %PUT Statement
Text is converted to uppercase unless enclosed in quotes	Text is not converted to uppercase
Use two quotes to produce one quote in text	Two quotes are printed as two quotes; to produce one quote in text, enclose the text in the %STR or %NRSTR function and code the text quote as %'
Quotes surrounding a string are not printed	Quotes surrounding a string are printed

See "SAS Macro Language" for a description of the macro language %PUT statement.

RUN Statement

Operating systems: All

The RUN statement causes the previously entered SAS step to be executed. The statement is often used in display manager and interactive sessions where, without a RUN statement, a SAS step is executed only after the first statement in the next DATA or PROC step is entered. It can also be used in batch processing to end a DATA or PROC step.

The form of the RUN statement is

RUN [CANCEL | QUIT];

where

CANCEL
QUIT — causes the SAS System to end the current step without executing it.[1]

This example uses two RUN statements:

```
TITLE 'USING PROC MEANS FOR ANALYSIS';
PROC MEANS DATA=ANYDATA MIN MAX;
   VAR X Y Z;
RUN;
TITLE 'USING PROC UNIVARIATE FOR ANALYSIS';
PROC UNIVARIATE DATA=ANYDATA;
   VAR X Y Z;
RUN;
```

The first RUN statement causes the SAS System to execute the PROC MEANS step before reading the second TITLE statement. Without the first RUN statement, the SAS System executes the PROC MEANS step after it reads the PROC UNIVARIATE statement. In that case, the second TITLE statement replaces the first TITLE statement and the output of both procedures contains the title for UNIVARIATE procedure.

These examples illustrate the use of the CANCEL or QUIT option in an interactive SAS session:

Output 12.1 Run Statement with QUIT Option

```
  1? *example for aos/vs, primos, vms;
  2? data circle;
  3?    input radius;
  4?    c=2*4.13*radius;
  5?    *incorrect value for pi;
  6? run;
ERROR: NO CARDS OR INFILE STATEMENT.
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: THE DATA STATEMENT USED THE FOLLOWING COMPUTER RESOURCES -
      BUFFERED I/O                14      ELAPSED TIME     00:00:29.01
      DIRECT I/O                   8      CPU TIME         00:00:00.68
      PAGE FAULTS                395
  7? data circle;
  8?    input radius;
  9?    c=2*4.13*radius;
 10?    *incorrect value for pi;
 11? run quit;
 12?
```

```
  1? *example for cms, os, vm/pc, and vse;
  2? data circle;
  3?     input radius;
  4?     c=2*4.13*radius;
  5?     *incorrect value for pi;
  6? run;

ERROR: NO INFILE OR CARDS STATEMENT USED.
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: DATA SET WORK.CIRCLE HAS 0 OBSERVATIONS AND 2 VARIABLES. 933 OBS/TRK.
NOTE: THE DATA STATEMENT USED 1.78 SECONDS AND 476K.

  7? data circle;
  8?     input radius;
  9?     c=2*4.13*radius;
 10?     *incorrect value for pi;
 11? run quit;

ERROR: NO INFILE OR CARDS STATEMENT USED.
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF CANCEL/QUIT.
NOTE: THE DATA STATEMENT USED 1.65 SECONDS AND 472K.

 12?
```

In each example SAS attempts to execute the first DATA step but not the second one.

Note

1. **AOS/VS, PRIMOS, and VMS:** SAS does not print messages associated with the step.

   CMS, OS, VM/PC, and VSE: SAS prints any messages associated with the step.

%RUN Statement

Operating systems: All

The %RUN statement can be used anywhere in a SAS job (except after a CARDS or PARMCARDS statement) to end source statements following a %INCLUDE * statement. The %RUN statement causes the SAS System to stop reading input from the terminal (including subsequent SAS statements on the same line) and resume reading from the previous input source.

See the %INCLUDE statement description for more information.

The form of the %RUN statement is

%RUN;

SKIP Statement

Operating systems: All

You can use the SKIP statement to skip lines on the SAS log.[1]

The form of the SKIP statement is

SKIP *n*;

where

n is the number of lines that you want to skip on the log. If the number specified is greater than the number of lines remaining on the page, the SAS System treats the SKIP statement like a PAGE statement and skips to the top of the next page. If *n* is omitted, the SAS System skips one line on the log.

Note

1. **AOS/VS, PRIMOS, and VMS**: The SKIP statement is listed on the log after the skipped lines.

CMS, OS, VM/PC, and VSE: The SKIP statement is not listed on the log.

TITLE Statement

Operating systems: All

Introduction

The TITLE statement is used to specify up to ten title lines to be printed on the SAS print file and other SAS output.

The form of the TITLE statement is

TITLE[*n*] [*title*];

where

n immediately follows the word TITLE, with no intervening blank, to specify the number of the title line. For example the statement

```
TITLE3 'THIS IS THE THIRD TITLE LINE';
```

specifies a title for the third line from the top of the page. *N* can range from 1 to 10. For the first title line, either TITLE or TITLE1 may be specified.

Omitting some values of *n* indicates that those lines are to be blank. For example, if TITLE1 and TITLE3 statements appear, a blank line appears between the two titles on the SAS output.

'title' gives a title of up to 132 characters that you want printed. If a title longer than the current line size is specified, it is truncated to the line length. Enclose the title in quotes.[1]

Here are examples of valid TITLE statements:

```
TITLE 'FIRST DRAFT';
TITLE2 'YEAR''S END REPORT';
TITLE5 "PROJECTED SALES FIGURES";
```

Once you specify a title for a line, it is used for all subsequent output until you cancel the title or define another title for that line. A TITLE statement for a given line cancels the previous TITLE statement for that line and for all lines with larger *n* values. To cancel all existing titles, specify

```
TITLE;
```

To suppress the *n*th and later titles, use

```
TITLEn;
```

You can insert SAS/GRAPH commands into TITLE statements. If the TITLE statement is used with a non-graphics procedure, the commands are ignored. For example:

```
TITLE C=RED H=2 F=COMPLEX 'THIS IS A TITLE';
```

If this statement is executed with a procedure other than a graphics procedure, the SAS/GRAPH commands are ignored and only the quoted string appears in the title. If the statement is used with a graphics procedure that produces a graphic display, the commands are interpreted and the title is produced accordingly.

Associating a Title with a Particular Step

If you want a title associated with a given PROC step, include the title

- after a RUN statement for the previous step, if one is present

or

- anywhere after the PROC statement and before the next DATA, PROC, or RUN statement.

For example, the statements

```
PROC PRINT;
   TITLE 'TITLE FOR FIRST PROC';
PROC MEANS;
```

print the title on the output for both the PRINT and the MEANS procedures. These statements

```
PROC PRINT;
RUN;
   TITLE 'TITLE FOR SECOND PROC';
PROC MEANS;
RUN;
```

print the title only on the PROC MEANS output. These statements

```
PROC PRINT;
PROC MEANS;
   TITLE 'TITLE FOR SECOND PROC';
```

also print the title only on the output pages from MEANS. See the **RUN Statement** earlier in this chapter for additional examples of associating titles with particular PROC steps.

Customizing Titles

You can use macro variables and macros to change the information in TITLE statements. If the title is enclosed in double quotes (quotation marks), the text indicated is substituted into the title; if the title is enclosed in single quotes (apostrophes), the text is not substituted. For example, the statements

```
%LET PERIOD=JANUARY AND FEBRUARY;
TITLE "SALES DURING &PERIOD";
```

produce the title

```
SALES DURING JANUARY AND FEBRUARY
```

See "SAS Macro Language" for information on macro variables.

Note

1. **CMS, OS, VM/PC, and VSE**: If the NOTEXT82 system option is in effect, you must enclose the title in quotes. If the TEXT82 system option is in effect, the title does not have to be in quotes; however, the SAS System prints a warning message.

TSO Statement

Operating system: OS

The TSO statement is used to execute TSO commands during a SAS session under TSO. The TSO statement is treated as a comment in OS batch jobs and under other operating systems.

The form of the TSO statement is

TSO [*TSOcommand*]**;**

where

TSOcommand is any TSO command or CLIST except the TEST, LOGON, or LOGOFF commands, any command which must be given control in an authorized state, or a CLIST containing the ATTN statement from the CLIST language. You can specify any TSO command up to 200 characters in length; do **not** enter a TSO continuation symbol for a command longer than one line.

The SAS System executes the TSO statement as soon as it is encountered. The TSO statement **must** end in a semicolon. For example, the TSO statement

```
TSO ALLOCATE FILE(IN) DATASET('USERID.MY.SASDATA');
```

issues an ALLOCATE command for the data set named USERID.MY.SASDATA without exiting from the SAS session.

If you omit the TSO command, as in the statement

```
TSO;
```

SAS enters TSO submode and prompts you for TSO commands. Commands entered in TSO submode are not processed as SAS statements and can be any length; if the command is longer than one line, you **must** enter a TSO continuation symbol. To return to the SAS session, enter either of the TSO subcommands

```
RETURN
```

or

```
END
```

Any characters following the RETURN or END subcommand are ignored. An END command appearing within a CLIST terminates the command procedure without ending TSO submode.

Here is an example:

Output 12.2 TSO Submode

```
 1? SAS statements
 ...
 11? tso;
TSO SUBMODE, ENTER "RETURN" OR "END" TO RETURN TO SAS
TSO
submit misc(compress)
JOB COMPRESS(JOB01931) SUBMITTED
TSO
status compress
JOB COMPRESS(JOB01931) ON OUTPUT QUEUE
TSO
return
 12? proc print; run;
```

See the *SAS Companion for OS Operating Systems and TSO* for more information.

X Statement

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, and VMS

Introduction

You can use the X statement to issue an operating system command from within a SAS session. The SAS System executes the X statement as soon as you enter it.

The form of the X statement is

X [*system*] ['[*command*]'];

where

system specifies the name of the operating system to which the command is directed. If the operating system under which the statement is issued is not *system*, the statement is treated as a comment. Values for *system* are AOSVS (note the absence of the slash), PRIMOS, and VMS.

Operating systems: AOS/VS, PRIMOS, and VMS

command specifies the operating system command. Omitting *command* puts you into the operating system's submode.[1]

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, and VMS

The commands you can issue with the X statement depend on your operating system. They are described briefly below; see the SAS Companion for your operating system for details.

Usage Notes for Operating System Commands

AOS/VS You can use the X statement to issue AOS/VS CLI commands. You can execute some CLI commands with the X statement only if you issue them as part of a string of commands. That is, you must decide which commands you want to execute and then enter them all with the same X statement using this format:

```
X 'command;command;command';
```

The following commands can be entered only as part of a string of commands:

ASSIGN	DISCONNECT
CHECKTERMS	LISTFILE
CLASS1	PREFIX
CLASS2	PREVIOUS
CONNECT	PROMPT
DATAFILE	PUSH
DEASSIGN	POP
DEFACL	SEARCHLIST
DIRECTORY	STRING

All AOS/VS commands given in the X statement are treated as comments in batch mode.

CMS and VM/PC The X statement works like the CMS statement. See the **CMS Statement** earlier in this chapter for information.

OS The X statement works like the TSO statement. See the **TSO Statement** earlier in this chapter for more information.

PRIMOS You can issue the internal PRIMOS commands listed below with the X statement:

ABBREV	INPUT	PRERR
ADD_REMOTE_ID	LD	RDY
ASRCWD	LISTING	REN
ASSIGN	LIST_ACCESS	RESTOR
ATTACH	LIST_GROUP	RESUME
BINARY	LIST_PRIORITY_ACCESS	RLS
CHANGE_PASSWORD	LIST_QUOTA	RSTERM
CLOSE	LIST_REMOTE_ID	SAVE
CNAME	LIST_VAR	SET_ACCESS
COMINPUT	LOGIN	SET_QUOTA
COMOUTPUT	LOGOUT	SET_VAR
COPY	LON	START
CPL	MESSAGE	STATUS
CREATE	NETLINK	SVCSW
DATE	OPEN	TIME
DEFINE_GVAR	ORIGIN	TYPE
DELAY	PASSWORD	UNASSIGN
DELETE_VAR	PHANTOM	USERS
DMSTK	PM	VRTSSW
EDIT_ACCESS	INPUT	

VMS The following VMS commands can be used in display manager, interactive, noninteractive, or batch processing:

ALLOCATE
ASSIGN
DEALLOCATE
DEASSIGN
DEFINE
DISMOUNT
MOUNT
SET DEFAULT
SET UIC

The following commands are accepted but ignored:

LOGOUT
SET ACCOUNTING
SET RMS_DEFAULT/PROCESS
SET WORKING_SET
SET COMMAND
SET PROCESS

All other commands can be used in display manager, interactive, or noninteractive (not batch) processing.

Note

1. **AOS/VS and VMS**: *command* must be enclosed in quotes. To enter the operating system's submode, use quotes without a command, as in

```
x '';
```

CMS and VM/PC: Quotes around the command are optional. The operating system must receive the command in uppercase. Therefore, if you enclose the command in quotes, either the system option CAPS must be in effect or you must enter the command in uppercase.

OS: Quotes around the command are optional.

PRIMOS: *command* is required and must be enclosed in quotes.

Chapter 13

SAS® Log and Procedure Output

The SAS System produces printed output in the form of the SAS log, a description of the SAS job or terminal session, and in the printed results of SAS procedures. This chapter discusses these forms of printed SAS output and how you can tailor them to meet your specific needs. The last section describes SAS errors and error messages.

Important: The SAS Display Manager provides another method of displaying the SAS log and procedure output. Refer to the "SAS Display Manager" chapter for details.

SAS LOG

The SAS log includes information about the processing of the SAS job: what statements were executed; what data sets were created and how many variables and observations they contain; how much time and memory each step in the job required; and, for batch jobs, the page number for locating the printed results of each procedure that was executed. The log is also used by some of the SAS procedures that perform SAS or utility functions, such as DATASETS, SOURCE, and TAPECOPY. Messages are written on the log by certain PUT statements.[1]

The SAS log is necessary and important documentation that gives a journal of the processing, and it can help you solve problems that arise during the job.

Destination of the Log

The destination of the log depends on the mode of execution you use, on the operating system, and on the setting of SAS system options.[2] Typically, the log destination is as follows by default:

- For interactive full-screen sessions, the log appears on the log screen of SAS Display Manager.
- For interactive line-mode sessions, the log appears on your terminal screen as you enter statements.
- For noninteractive SAS programs, the log either appears on the terminal screen as each step executes or it is written to a disk file, depending on your operating system.
- For batch SAS jobs, the log is routed to a line printer or to a disk file, depending on your operating system.

Be sure to see Note 2 at the end of this chapter for details on output destinations for your operating system and for information on overriding default destinations. Refer also to the SAS Companion or Technical Report that pertains to your operating system for information on output destinations.

Structure of the SAS Log

Each line in your SAS job containing SAS statements is printed and numbered on the log; for example, the small number 1 printed to the left of the OPTIONS statement in the example below means that it was the first line in the job.

Interspersed with your SAS statements are messages from the SAS System. These messages sometimes begin with the word NOTE, the word ERROR, or an error number. They sometimes refer to a SAS statement by its line number on the log.

Output 13.1 and **Output 13.2** are two sample SAS logs that were created by running identical SAS jobs on two different operating systems. The numbered items shown on the logs are explained in the numbered list below. Note that not all the items appear on SAS logs for jobs run under all operating systems.

Output 13.1 SAS Log for a Job Run under OS

```
1         SAS(R) LOG   OS SAS 5.XX      ❷  MVS/XA JOB SASTEST  STEP SASTEST       ❶  16:38 TUESDAY, JANUARY 8, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC. ❹     (XXXXXXXX).  ❸

NOTE: SAS OPTIONS SPECIFIED ARE:      ❺
❼  LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1          DATA HTWT;
2  ❽         INPUT NAME $ 1-10 SEX $ 12 AGE 14-15 HEIGHT 17-18 WEIGHT 20-22;
3            LIST;
4            CARDS;


RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

5          ALFRED     M 14 69 112
6          ALICE      F 13 56  84
```

(continued on next page)

```
(continued from previous page)

7          BARBARA    F 14 62 102
8          BERNADETTE F 13 65  98
9          HENRY      M 14 63 102
10         JAMES      M 12 57  83
11         JANE       F 12 59  84
12         JANET      F 15 62 112
13         JEFFREY    M 13 62  84
14         JOHN       M 12 59  99
15         JOYCE      F 11 51  50
16         JUDY       F 14 64  90
17         LOUISE     F 12 56  77
18         MARY       F 15 66 112
19         PHILIP     M 16 72 150
20         ROBERT     M 12 64 128
21         RONALD     M 15 67 133
22         THOMAS     M 11 57  85
23         WILLIAM    M 15 66 112
NOTE: DATA SET WORK.HTWT HAS 19 OBSERVATIONS AND 5 VARIABLES. 488 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.98 SECONDS AND 372K.
(11)
24
24            PROC PRINT;
25
NOTE: THE PROCEDURE PRINT USED 0.89 SECONDS AND 472K AND PRINTED PAGE 1.
(11)
25            PROC PLOT;
26               PLOT HEIGHT*WEIGHT=SEX;
NOTE: THE PROCEDURE PLOT USED 0.89 SECONDS AND 468K AND PRINTED PAGE 2.
NOTE: SAS USED 472K MEMORY.
(11)
```

```
2          SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST          16:38 TUESDAY, JANUARY 8, 1985

(12) NOTE: SAS INSTITUTE INC.
       SAS CIRCLE
       PO BOX 8000
       CARY, N.C. 27511-8000
```

Output 13.2 SAS Log for a Job Run under VMS

```
                                          S A S  L O G     VMS SAS 5.XX          (1)  9:37 TUESDAY, JANUARY 8, 1985  1

Copyright (c) 1985 SAS Institute Inc., Cary, N. C. 27511, U. S. A.
NOTE: VMS Version of SAS Release 5.XX at SAS INSTITUTE INC.   (XXXXXXXX).  (3) (4) (5)
NOTE: LICENSED CPUID MODEL = XX/XXX, SERIAL = XXXXXXXX.  (6)
(8)    1 DATA HTWT;
       2   INPUT NAME $ 1-10 SEX $ 12 AGE 14-15 HEIGHT 17-18 WEIGHT 20-22;
       3 LIST;
       4 CARDS;
RULE:--- ----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8---+----9----+----0----+----1----+----2
   5 ALFRED     M 14 69 112
   6 ALICE      F 13 56  84
   7 BARBARA    F 14 62 102
   8 BERNADETTE F 13 65  98
   9 HENRY      M 14 63 102
  10 JAMES      M 12 57  83
  11 JANE       F 12 59  84
  12 JANET      F 15 62 112
  13 JEFFREY    M 13 62  84
  14 JOHN       M 12 59  99
  15 JOYCE      F 11 51  50
  16 JUDY       F 14 64  90
  17 LOUISE     F 12 56  77
  18 MARY       F 15 66 112
  19 PHILIP     M 16 72 150
  20 ROBERT     M 12 64 128
  21 RONALD     M 15 67 133
  22 THOMAS     M 11 57  85
  23 WILLIAM    M 15 66 112
NOTE: DATA SET WORK.HTWT HAS 19 OBSERVATIONS AND 5 VARIABLES.
NOTE: THE DATA STEP USED THE FOLLOWING COMPUTER RESOURCES -
      BUFFERED I/O            8      ELAPSED TIME    00:00:04.09
(11)  DIRECT I/O              8      CPU TIME        00:00:02.96
      PAGE FAULTS           801
   24 PROC PRINT;

                                                                                                  (continued on next page)
```

```
(continued from previous page)

NOTE: THE PROCEDURE PRINT USED THE FOLLOWING COMPUTER RESOURCES -
      BUFFERED I/O           10      ELAPSED TIME    00:00:02.71
(11)  DIRECT I/O              6      CPU TIME        00:00:01.35
      PAGE FAULTS           374
    25 PROC PLOT;
    26   PLOT HEIGHT*WEIGHT=SEX;
NOTE: THE PROCEDURE PLOT USED THE FOLLOWING COMPUTER RESOURCES -
      BUFFERED I/O           6      ELAPSED TIME    00:00:02.33
(11)  DIRECT I/O              6      CPU TIME        00:00:01.59
      PAGE FAULTS           78
NOTE: SAS USED 472K MEMORY.
(12)
NOTE: SAS INSTITUTE INC., SAS CIRCLE, BOX 8000, CARY, N. C., 27511-8000
(13)
```

1. Date and time the job was run.
2. Name of the job taken from the JOB statement in the job control language of your batch job.
3. The SAS System release used to run this job.
4. Name of the computer installation where the job was run.
5. The computer installation site number. (The site number is used by SAS Institute to identify the installation given in 4.)
6. CPU serial number for your installation.
7. SAS system options specified when you invoke SAS. (This line is printed when the OPLIST system option is in effect. See the "SAS System Options" chapter for more information.)
8. SAS statements for each DATA or PROC step in the job.
9. For each external file of data read or written in a DATA step, information about the file, including the number of lines read or written.
10. For each SAS data set created, its name, the number of observations and variables, and the number of observations that can be stored in one track of disk space.
11. Computer time and memory used by the step; for a step that produces printed output, the pages printed.
12. Maximum amount of memory actually used by the steps in the job.
13. The name and address of SAS Institute Inc. When you see this note, you know that your job ran to completion. If you cannot find the note, check the messages from the computer's operating system to see if the job needed more time or ran into a system problem.

Writing on the Log

In addition to the information that SAS automatically writes to the log, you can have other information printed there. You can tell SAS to write to the log in four ways: the LIST statement automatically prints the current data line on the log; PUT statements can be directed to the log; %PUT statements always write on the log.[3] **Output 13.3** is the log of a SAS job that uses a LIST statement and a PUT statement directed to the log. (Note that in this DATA step FILE LOG is assumed since no FILE statement is present.)

When SAS encounters a LIST statement, it flags the current data line and prints it on the SAS log before returning to the beginning of the DATA step for a new execution. On the other hand, the PUT statement causes SAS to print immediately. Thus, for observations with X and Y equal, the PUT statement prints before the LIST statement writes the data line.

Output 13.3 Writing on the Log

```
1          SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST          14:56 MONDAY, JANUARY 14, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1           DATA ONE;
2             INPUT ID $ X Y;
3             LIST;
4             IF X=Y THEN PUT 'X AND Y ARE EQUAL ' X=;
5             CARDS;


RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

6           A01 1.2 3
7           A21 2.4 4
X AND Y ARE EQUAL X=2
8           A02 2   2
9           B01 3   0
X AND Y ARE EQUAL X=3
10          B21 3   3
11          B02 1   .
12          C01 .   5
NOTE: DATA SET WORK.ONE HAS 7 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.97 SECONDS AND 372K.

13           ;
14          PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 0.89 SECONDS AND 472K AND PRINTED PAGE 1.
NOTE: SAS USED 472K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Suppressing All or Part of the Log

When you have large SAS programs that you run on a regular basis without changes, you may want to suppress the listing of your SAS statements on the log. You can use the system option NOSOURCE in the control language when you invoke SAS or in an OPTIONS statement at the beginning of your job to suppress these lines.

Sometimes you may want to suppress the notes that SAS prints on the log. You can use the system option NONOTES to prevent SAS from printing any of the messages beginning with NOTE. **Do not** use this option until your program is error free. The notes that SAS prints are required for debugging.

When data errors occur in your SAS job, SAS prints error messages for up to *n* errors where *n* is the value of the system option ERRORS=. Normally, the default value for ERRORS= is 20, and SAS prints messages for up to 20 data errors. You can set the value of ERRORS= to another number to specify the maximum number of error messages you want printed on your log. If ERRORS=0, no error messages are printed. (Note that the ? or ?? message modifiers in the INPUT statement also affect the printing of error messages. See the description of the INPUT statement for more information.)

For more information about these and other system options, see the chapter "SAS System Options."

Skipping Lines and Pages

You can use the SKIP and PAGE statements to control the appearance of the SAS log. See SKIP and PAGE in "SAS Statements Used Anywhere" for more information.

PRINTED RESULTS OF SAS PROCEDURES

The pages after the SAS log contain the results of PROC steps as well as reports printed by DATA steps using the PUT and FILE statements routed to a PRINT file. These results and reports appear in the same order as the corresponding DATA and PROC steps appeared in the job, and notes on the log tell which output pages were produced by which DATA or PROC step.

Several examples of reports written in DATA steps with PUT and FILE statements are shown in "Data Step Applications."

Each SAS procedure produces a different form of output. Consult the procedure descriptions in this user's guide and the *SAS User's Guide: Statistics* for examples of output from SAS procedures. You can tailor SAS printouts using certain SAS statements and system options.

Procedure Output Destination

The destination of your procedure output depends on the mode of execution you use, on the operating system, and on the setting of SAS system options. Typically, the destination of procedure output is as follows by default:

- For interactive full-screen sessions, procedure output appears on the output screen of SAS Display Manager.
- For interactive line-mode sessions, procedure output appears on your terminal screen as each step executes.
- For noninteractive SAS programs, procedure output either appears on the terminal screen after the job executes or it is written to a disk file, depending on your operating system.
- For batch SAS jobs, procedure output is routed to a line printer or to a disk file, depending on your operating system.

Be sure to see Note 2 at the end of this chapter for details on output destinations for your operating system and for information on overriding default destinations. Refer also to the SAS Companion or Technical Report pertaining to your operating system for information on output destinations.

Titles

SAS prints the title

```
SAS
```

at the top of each page of output unless you specify your own titles with one or more TITLE statements. See the TITLE statement in "SAS Statements Used Anywhere" for more information.

Printing Values

You have some control over the way SAS prints values in procedure output. For example, by default SAS prints a single dot (.) as a missing value for a numeric variable. To have SAS print some other character for a missing value, use the SAS system option MISSING= in the control language when you invoke SAS or in an OPTIONS statement. For example, if you specify

```
OPTIONS MISSING='B';
```

when the value of a numeric variable is missing, the letter B is printed instead of a single period.

You can use a FORMAT statement to associate formats with variables. This format is then used to print the values of variables. The SAS formats available are described in "SAS Informats and Formats."

You can define your own formats using the FORMAT procedure. This procedure gives you a great deal of flexibility in printing the results of procedures.

See descriptions of the FORMAT statement and the description of the FORMAT procedure for more information.

Printing Variable Names

For most SAS procedures, if a LABEL statement has been specified in the DATA step used to create the data set being analyzed or in the current PROC step, the LABEL rather than the variable name is printed. The LABEL can be up to forty characters long, thus giving a more descriptive name to the variable. See the LABEL statement descriptions for more information.

Page and Line Sizes

The default number of lines per page (page size) and characters per line (line size) are determined by the values of the SAS system options PAGESIZE= (or PS) and LINESIZE= (or LS). You can reset these options to your own values with the OPTIONS statement.

For example, suppose you are planning to photo-reduce your output to fit on an 8.5" x 11" page. Instead of the default values of 60 lines per page and 132 characters per line, you will be able to fit 76 lines per page but only 100 characters per line. If you use the OPTIONS statement below in your SAS job, output printed uses 76 lines per page and 100 characters per line.

```
OPTIONS PAGESIZE=76 LINESIZE=100;
```

The values you use for LINESIZE= and PAGESIZE= can affect the output produced by some SAS procedures. For example, the default set of descriptive statistics printed by the MEANS procedure is dependent on the line size in effect at the time. The appearance of the output from procedures changes slightly for different line sizes. When running SAS interactively at a terminal, you can control the width of the line with the system option TLS (terminal line size).[4] SAS provides a default TLS value appropriate for your terminal, but you can reset the TLS value using the OPTIONS statement.

Page Numbering

SAS numbers the output pages in batch jobs at the top of the page, beginning with page 1.

If you do not want page numbers printed on your output, use the SAS system option NONUMBER. For example, you could specify this OPTIONS statement:

```
OPTIONS NONUMBER;
```

as the first statement in your SAS job to suppress page numbering.

Date and Time

The date and time that the job was run appear on the printout. These values become important when you are running a program several times.

If you do not want the time and date values to appear, you can use the system option NODATE. For example, this OPTIONS statement

```
OPTIONS NODATE;
```

causes SAS to leave off the current date and time from each page of output.

Centering Output

SAS normally centers titles and procedure output on the pages in batch jobs. If you want the output left aligned rather than centered, you can use the system option NOCENTER:

```
OPTIONS NOCENTER;
```

See the OPTIONS statement in "SAS Statements Used Anywhere" for more information about SAS systems options.

ERRORS

SAS can detect three kinds of errors: syntax errors, programming errors, and data errors. SAS finds syntax errors as it compiles each SAS step before the statements are executed. Data errors and some programming errors are discovered when your SAS job is being executed.

Syntax Errors

If you misspell a SAS keyword, forget a semicolon, or make similar mistakes, SAS prints the word ERROR followed by an error number. A message explaining the error is printed at the end of the DATA or PROC step.

Some errors are explained fully by the message that SAS prints. For example, in **Output 13.4**, the INFILE statement in the DATA step references a fileref (DDname) that has not been previously defined. In the PROC PRINT step, the VAR statement specifies the name of a variable that is not in the data set being analyzed.

Output 13.4 A Syntax Error

```
1          SAS(R) LOG   OS SAS 5.XX          VS2/MVS JOB SASTEST  STEP SASTEST          15:16 MONDAY, JANUARY 14, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 2.15 SECONDS.
1           DATA ONE;
2             INFILE IN2;
                     526
3             INPUT ID $ X Y;


ERROR 526: DDNAME NOT FOUND.

NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: SAS SET OPTION OBS=0 AND WILL CONTINUE TO CHECK STATEMENTS.
      THIS MAY CAUSE NOTE: NO OBSERVATIONS IN DATA SET.
NOTE: DATA SET WORK.ONE HAS 0 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 2.57 SECONDS AND 368K.

ERROR: VARIABLE Z NOT FOUND.
4           PROC PRINT;
5             VAR X Y Z;
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF CANCEL/QUIT.
NOTE: THE PROCEDURE PRINT USED 2.44 SECONDS AND 468K.
NOTE: SAS USED 468K MEMORY.

ERROR: ERRORS ON PAGES 1.

NOTE: SAS INSTITUTE INC.
```

(continued on next page)

(continued from previous page)

```
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The error number is printed so that the first digit of the number is below the first character of the word in error. The number 526 at the end of the step explains the error.

Other error messages are not as easy to interpret as the example above. For example, because SAS statements are free format and can begin and end anywhere, when you fail to end a SAS statement with a semicolon, SAS does not always detect the error. Here is an example:

```
DATA A;
  INPUT X Y
  Z=X + Y;
  LIST;
  CARDS;
1 2
2 3
4 5
6 7
;
PROC PRINT;
   TITLE 'THE EFFECT OF A MISSING SEMICOLON';
```

The log from the program above shows that the missing semicolon in the INPUT statement is not detected as an error by SAS. (See **Output 13.5**.)

Output 13.5 A Syntax Error SAS Did Not Detect

```
1          SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST            16:18 MONDAY, JANUARY 14, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.       (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1           DATA A;
2             INPUT X Y
3             Z=X + Y;
4             LIST;
5             CARDS;


RULE:       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

7           2 3
9           6 7
NOTE: SAS WENT TO A NEW LINE WHEN INPUT STATEMENT
      REACHED PAST THE END OF A LINE.
NOTE: DATA SET WORK.A HAS 2 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.97 SECONDS AND 372K.

10          ;
11          PROC PRINT;
12            TITLE 'THE EFFECT OF A MISSING SEMICOLON';
```

(continued on next page)

```
(continued from previous page)

NOTE: THE PROCEDURE PRINT USED 0.89 SECONDS AND 472K AND PRINTED PAGE 1.
NOTE: SAS USED 472K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The assignment statement is considered part of the INPUT statement: Z= is treated as named input, X tells SAS to read a value for X (this value replaces the first X value read), and +Y is a valid pointer control telling SAS to skip the next Y columns.

The message

```
NOTE: SAS WENT TO A NEW LINE WHEN INPUT STATEMENT REACHED
      PAST THE END OF A LINE.
```

is often an indication that your data lines are not being read as you expected. When the LIST statement is executed, the current record is the second in each group of two lines. The PRINT procedure prints the data set created (see **Output 13.6**).

Output 13.6 The Output from PROC PRINT Reveals the Error

```
                          THE EFFECT OF A MISSING SEMICOLON                1

                              OBS    X    Y    Z

                               1     2    2    .
                               2     6    5    .
```

Below is a DATA step where a semicolon was left off the DATA statement. Again, SAS did not detect the error (see **Output 13.7**).

```
DATA A
  INPUT X Y;
  Z= X + Y;
  LIST;
  CARDS;
1 2
3 4
5 6
7 8
;
PROC PRINT;
   TITLE 'A MISSING SEMICOLON IN THE DATA STATEMENT';
```

Output 13.7 SAS Did Not Detect the Missing Semicolon

```
1         SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST          17:19 MONDAY, JANUARY 14, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1           DATA A
2             INPUT X Y;
3             Z=X + Y;
4             LIST;
5             CARDS;

NOTE: THE VARIABLE X IS UNINITIALIZED.
NOTE: THE VARIABLE Y IS UNINITIALIZED.
NOTE: MISSING VALUES WERE GENERATED AS A RESULT OF PERFORMING
      AN OPERATION ON MISSING VALUES.
      EACH PLACE IS GIVEN BY: (NUMBER OF TIMES) AT (LINE):(COLUMN).

      1 AT 3:5

NOTE: DATA SET WORK.A HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: DATA SET WORK.INPUT HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: DATA SET WORK.X HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: DATA SET WORK.Y HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 1.02 SECONDS AND 380K.

10          ;
11          PROC PRINT;
12            TITLE 'A MISSING SEMICOLON IN THE DATA STATEMENT';
NOTE: THE PROCEDURE PRINT USED 0.89 SECONDS AND 472K AND PRINTED PAGE 1.
NOTE: SAS USED 472K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The INPUT statement was assumed to be part of the DATA statement. Thus, SAS was prepared to build four data sets: A, INPUT, X, and Y. One observation was written to each data set when the DATA step was executed.

The messages on the SAS log indicate that four data sets were created—WORK.A, WORK.INPUT, WORK.X, and WORK.Y—each with one observation. This is your clue that an error occurred. Since no INPUT statement was executed, X and Y were assumed to be missing in the assignment statement, and the LIST statement had no record to write. The resulting PROC PRINT is shown in **Output 13.8**.

Output 13.8 PROC PRINT Shows the Results of the Missing Semicolon

```
                    A MISSING SEMICOLON IN THE DATA STATEMENT                    1

                              OBS    Z    X    Y

                               1     .    .    .
```

When SAS finds a syntax error, it does not run the step in which the error occurred. To warn you, it prints:

```
NOTE:  SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
```

Syntax Check Mode

If you were executing PROC DATASETS or an external procedure or creating a SAS data set and the step contains a syntax error, then SAS prints:

```
NOTE:  SAS SET OPTION OBS=0 AND WILL CONTINUE TO CHECK STATEMENTS.
       THIS MAY CAUSE NOTE:  NO OBSERVATIONS IN DATA SET.
```

In an effort to continue processing, SAS executes later steps in the job with OBS=0 if

- a SAS data set is created in the step
- no SAS data sets are used as input to the step.

If either of these conditions holds, the step is executed. However, permanent SAS data sets are not replaced.

When OBS=0, SAS reads zero observations from data sets or external files. Thus, any data set created in the step may contain zero observations. When a SAS procedure tries to analyze one of these data sets, it prints the message

```
NOTE:  NO OBSERVATIONS IN DATA SET.
```

and stops.

Programming Errors

The execution-time errors that cause your SAS job to fail are called *programming errors*. For example, if your program processes an array and SAS encounters a value of the array's subscript that is out of range, SAS prints an error message and stops.

Below is an example. The DATA step defines an array with ten elements. Variable I contains the value of the subscript for the current execution of the DATA step. The value of I is read with an INPUT statement before the array is processed in a programming statement. A miscoded I value results in an error message:

```
DATA A;
  ARRAY ALL(*) X1-X10;
  INPUT I MEASURE;
    IF MEASURE>0 THEN ALL(I)=MEASURE;
    CARDS;
1 1.5
. 3
2 4.5
;
PROC PRINT;
```

See **Output 13.9.**

Output 13.9. A Miscoded Value Causes an Error Message to Be Printed

```
1          SAS(R) LOG   OS SAS 5.XX          VS2/MVS JOB SASTEST  STEP SASTEST          13:58 TUESDAY, JANUARY 15, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4
```

(continued on next page)

(continued from previous page)

```
NOTE: THE INITIALIZATION PHASE USED 2.18 SECONDS.
1          DATA A;
2            ARRAY ALL{*} X1-X10;
3            INPUT I MEASURE;
4            IF MEASURE>0 THEN ALL{I}=MEASURE;
5            CARDS;

ERROR: EXPLICIT ARRAY SUBSCRIPT OUT OF RANGE AT LINE 4 COLUMN 21.

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

7          . 3
X1=. X2=. X3=. X4=. X5=. X6=. X7=. X8=. X9=. X10=. I=. MEASURE=3 _ERROR_=1 _N_=2
NOTE: SAS SET OPTION OBS=0 AND WILL CONTINUE TO CHECK STATEMENTS.
      THIS MAY CAUSE NOTE: NO OBSERVATIONS IN DATA SET.
NOTE: DATA SET WORK.A HAS 1 OBSERVATIONS AND 12 VARIABLES. 190 OBS/TRK.
NOTE: THE DATA STATEMENT USED 3.04 SECONDS AND 380K.

9           ;
10         PROC PRINT;
NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF CANCEL/QUIT.
NOTE: THE PROCEDURE PRINT USED 2.55 SECONDS AND 480K.
NOTE: SAS USED 480K MEMORY.

ERROR: ERRORS ON PAGES 1.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Data Errors and Other Warning Messages

Like programming errors, data errors are detected during the execution of the job. However, unlike programming errors, data errors do not cause execution to stop. SAS warns you of the data errors on the SAS log.

The DATA step below reads seven data lines:

```
OPTIONS LS=80;
DATA ONE;
  INPUT ID $ X Y;
  Z=X/Y;
  LG=LOG(X);
  CARDS;
a01 1.2 3
a21 2.4 4
a02 0   2
b01 3   0
b21 A   3
b02 1   .
c01 .   5
;
PROC PRINT;
```

The SAS log produced when the job is run in batch contains several warning messages; however, the job ran to completion. Notice that data lines following a CARDS statement are numbered along with the log statements (see **Output 13.10**).

Output 13.10 Warning Messages are Printed When SAS Encounters Data Errors

```
1         SAS(R) LOG   OS SAS 5.XX          VS2/MVS JOB SASTEST  STEP SASTEST          14:17 TUESDAY, JANUARY 15, 1985

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 2.19 SECONDS.
1           DATA ONE;
2             INPUT ID $ X Y;
3             Z=X/Y;
4             LG=LOG(X);
5             CARDS;

NOTE: ILLEGAL ARGUMENT TO FUNCTION AT LINE 4 COLUMN 6.

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

8         A02 0   2
ID=A02 X=0 Y=2 Z=0 LG=. _ERROR_=1 _N_=3
NOTE: DIVISION BY ZERO AT LINE 3 COLUMN 6.
9         B01 3   0
ID=B01 X=3 Y=0 Z=. LG=1.098612 _ERROR_=1 _N_=4
NOTE: INVALID DATA FOR X IN LINE 10 5-5.    2:14
10        B21 A   3
ID=B21 X=. Y=3 Z=. LG=. _ERROR_=1 _N_=5
NOTE: MATHEMATICAL OPERATIONS COULD NOT BE PERFORMED AT THE
      FOLLOWING PLACES. THE RESULT OF THESE OPERATIONS HAVE
      BEEN SET TO MISSING VALUES.
      EACH PLACE IS GIVEN BY: (NUMBER OF TIMES) AT (LINE):(COLUMN).

      1 AT 3:5   1 AT 4:6

NOTE: MISSING VALUES WERE GENERATED AS A RESULT OF PERFORMING
      AN OPERATION ON MISSING VALUES.
      EACH PLACE IS GIVEN BY: (NUMBER OF TIMES) AT (LINE):(COLUMN).

      3 AT 3:5   2 AT 4:6

NOTE: DATA SET WORK.ONE HAS 7 OBSERVATIONS AND 5 VARIABLES. 433 OBS/TRK.
NOTE: THE DATA STATEMENT USED 2.91 SECONDS AND 372K.

13         PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 2.50 SECONDS AND 476K AND PRINTED PAGE 1.
NOTE: SAS USED 476K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The first data error occurs when X is 0. Zero is an illegal argument to the LOG function. The error causes SAS to print several types of information. They are

- the warning message, which in this case is

  ```
  NOTE:  ILLEGAL ARGUMENT TO FUNCTION AT LINE 4 COLUMN 6.
  ```

- for the first error, the RULE, which gives the column numbers referred to in this and later messages.
- the current record in the input buffer (usually the line in error).
- the current values of all the variables.

  For each subsequent error that occurs while SAS is executing the DATA step, SAS prints the warning message, the line in error, and the current values of the variables.
- At the end of the step, SAS prints a summary of the operations that could not be performed because of invalid arguments or divisors. The number of times a missing value was the result of an operation is also given. The operations that resulted in missing values are summarized in the form:

  ```
  (NUMBER OF TIMES) AT (LINE):(COLUMN)
  ```

For example, the SAS log above summarizes the number of times that missing values were generated as a result of performing operations on missing values:

```
3 AT 3:5  2 AT 4:6
```

This note tells you that missing values resulted three times at line 3, column 5 of the SAS log—the location of the variable X in the assignment statement

```
Z=X/Y;
```

and two times at line 4, column 6 of the SAS log in the statement

```
LG=LOG(X);
```

where the LOG function appears.

NOTES

1. **CMS, OS, VM/PC, and VMS:** ERROR statements also write to the log.
2. **AOS/VS:** For noninteractive jobs, the log and procedure output are written to files in your current default directory. The files are called *filename*.LOG and *filename*.LIS, respectively, where *filename* is the name of the file containing the SAS program.

For batch jobs, the log and procedure output are again written to files named *filename*.LOG and *filename*.LIS, but the files are written to the directory containing the SAS source file, not to the current default directory.

The destination of output is controlled by the LTYPE, LDISK, LPRINT, PTYPE, PDISK, and PPRINT options. See the "SAS System Options" chapter for descriptions.

CMS and VM/PC: For noninteractive jobs, the log is written to a disk file with filetype SASLOG and filemode A by default. Procedure output is written to a disk file with filetype LISTING and filemode A by default.

The destination of output is controlled by the LTYPE, LDISK, LPRINT, PTYPE, PDISK, and PPRINT options. See the "SAS System Options" chapter for descriptions.

OS and VSE: For noninteractive jobs, the log and procedure output appears on the screen as each step executes. For batch jobs, output is routed to a printer. To alter these defaults, change the control language for the log file (FT11F001) and the procedure output file (FT12001) as described in the SAS Companion for your operating system.

PRIMOS: For noninteractive jobs, the log and procedure output are written to files in your current default directory. The files are called *filename*.LOG and *filename*.LIST, respectively, where *filename* is the name of the file containing the SAS program.

For batch jobs, the log and procedure output are again written to files named *filename*.LOG and *filename*.LIST. These files are routed to your current attach point. that is, the directory to which you are attached at the time you submit the job.

The destination of output is controlled by the LTYPE, LDISK, LPRINT, PTYPE, PDISK, and PPRINT options. See the "SAS System Options" chapter for descriptions.

VMS: For noninteractive jobs, the log and procedure output are written to files in your current default directory. The files are called *filename*.LOG and *filename*.LIS, respectively, where *filename* is the name of the file containing the SAS program. Use the VMS command TYPE to display these files at your terminal and the command PRINT to route them to the printer.

For batch jobs, the log and procedure output are again written to files named *filename*.LOG and *filename*.LIS, but the files are written to your home directory, not to your current default directory.

The destination of output is controlled by the LTYPE, LDISK, LPRINT, PTYPE, PDISK, and PPRINT options. See the "SAS System Options" chapter for descriptions.

3. See note 1.

4. **VSE:** For ICCF under VSE, the LINESIZE from the INTERACTIVE option set is used.

Chapter 14

SAS® Display Manager

INTRODUCTION

The SAS Display Manager System is a full-screen facility that allows you to interact with all parts of your SAS job—program statements, log, and procedure output. It provides you with a full-screen editor for inputting and preparing your SAS statements and data, displays the SAS log created when you run a SAS job, and displays the output produced by your SAS program statements. Display manager has three primary screens:

- the program editor screen where you input, edit, and save SAS source files and submit SAS program statements
- the log screen where you can browse the SAS log
- the output screen where you can browse output from your SAS jobs.

You also have access to special screens, such as the function key definition screen, and to SAS HELP, an on-line help facility where you can browse information about the entire SAS System.

Screen 14.1 Program Editor/Log Screen

The display manager first shows you **Screen 14.1**. The bottom half is a program entry and edit area for your SAS statements; the top half of this split screen displays the SAS log.

Program Editor Screen

You type SAS statements in the program editor, review, make changes, and submit the statements for execution. In a SAS line-prompt session your program statements are submitted to the SAS System each time you enter a line; in a display manager session, you can delete and insert lines, enter, edit, and re-edit your SAS program statements and then submit them to SAS all at one time.

Log Screen

The top half of **Screen 14.1** is the log screen. Statements submitted to the SAS System from the program editor produce the SAS log that, together with any notes or error messages, is displayed in the display manager log screen. Except for entering commands on the command line, you cannot alter any text displayed in the log screen. As statements and messages build up in the SAS log, lines scroll off the top of the screen. Scrolling commands, which can be executed from the command line as well as with function keys, are available for browsing text in the log screen.

Output Screen

When you submit SAS program statements containing a PROC step that produces printed output, you can view that output in the output screen. The display manager displays the first page of the output and then allows you to see the rest of the output, one page at a time, or to skip to the last page of output. After the last page of requested output has been displayed, you can choose to return to the program editor/log screen or to browse the output in page-browse or line-browse mode. You can use function keys to scroll in page-browse mode; you can use function keys and execute command-line commands in line-browse mode.

TERMINALS, KEYS, AND COMMANDS

Function Keys

The SAS Display Manager System defines function keys for you to use to prepare your SAS program statements, to scroll information displayed on each screen, and to move from one screen to another. Some keys on your terminal keyboard, therefore, take on new meanings when you enter display manager. The default function key settings for display manager that were provided to your SAS site are described in this chapter. To set your own function keys, see the appendix on "Full-Screen Editing" for a discussion of the user profile facility.

Function keys on each screen The set of function keys that is available to you depends on the display manager screen with which you are working. In addition to entering commands directly on the command line or on line numbers, you can use function keys to execute commands. See the appendix on "Full-Screen Editing" for a discussion of

- using function keys to execute commands
- viewing the default settings

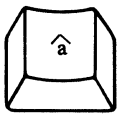
- altering the settings
- storing your altered settings.

Editing Keys

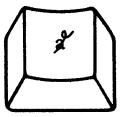
You can continue using some of your terminal's own editing keys with display manager, such as enter, insert, delete, and cursor movement keys. Ask someone knowledgeable at your site if you need help getting started with these keys.

The editing keys you use with display manager are shown below. You can use these same keys when editing program statements in the program editor or when entering commands on the command line of the program editor, log, or output screens. You can use these keys in all **unprotected fields**. In general, you choose an editing key to make a change to one character or one line of an unprotected field on the screen, whereas you choose a function key to search or scroll an entire file.

Note: the words or symbols on the editing keys described in **Figure 14.1** may not be the same as those on the terminal you are using.[1]



Press the INSERT key once to turn it on, and position the cursor where you want to insert a character. The next key you press inserts its character before the cursor position, shifting characters to the right to make room.*



The DELETE key erases the character either preceding the cursor position or in the current cursor position, depending on the terminal you are using.

The ERASE EOF (End-of-Field) key deletes or erases all the characters from the cursor position to the end of either a data entry line or, on some screens, the end of a field.

The HOME key moves the cursor to the first column of the first unprotected field on the screen, the entry area of the command line on most screens.**

The REFRESH key redisplays the contents of the screen line-by-line. It removes messages from the operating system and is especially useful for removing unwanted characters input since the last time you pressed ENTER or a function key.***

The NEW LINE key moves the cursor to the first unprotected field of the next line. Use the NEXT FIELD key to move the cursor from the line number to the data entry area.

---

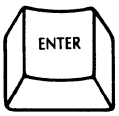
* **IBM users:** On a 3270 series terminal, press the RESET key to discontinue the INSERT.
Minicomputer users: Press the INSERT key again to discontinue the INSERT.

** **IBM users:** Press the ALT and Home keys.

*** **CMS, OS and VM/PC:** Press ALT PA2.
VSE: Press ALT CLEAR.

The PRIOR FIELD key moves the cursor to the previous unprotected field.

The NEXT FIELD key moves the cursor to the next unprotected field.



The ENTER key executes a command entered on a command line or on a line number. Do not confuse the ENTER key with the SUBMIT key. This key executes commands; the SUBMIT key submits SAS statements to the SAS System from the data lines of the program editor screen.

Figure 14.1 Editing Keys

Commands

Most of the commands you use with display manager are *command-line commands* entered on the command line of each screen. Notice in **Screen 14.1** that both screens in the program editor/log screen have a command line in the top left corner. You can also add a command line to the output screen.

Another type of command, called a *line command*, is used for editing SAS program statements and is available only in the program editor screen. See the appendix on "Full-Screen Editing" for a discussion of both kinds of commands, their execution, and command definitions. Commands you can use in display manager are listed according to screen in the sections that follow. Command definitions appear in the **Command Glossary** at the end of this chapter.

A COMPLETE EXAMPLE

This section uses an education example to go through a SAS display manager session step-by-step. This example illustrates a simple SAS program that you can use display manager to execute. You can enter your DATA and PROC statements in the program editor screen, use the log screen to view your SAS log, and use the output screen to view your output.

Suppose your company encourages employees to continue their education by taking night courses at a local college, weekend seminars, and so on. You have available for each person in your department:

- his or her name
- the number of credit hours he or she completed in the past year at the college
- the number of hours he or she spends in other educational activities
- the number of years he or she has been in your department.

You want to find out whether people who have been in your department for a short time participate more in continuing education than those with several years' seniority. You can:

1. create a data set, giving it a two-level name if you want to store it
2. identify your variables, the employee's name, the number of education hours spent in regular courses, the number of hours spent in special educational activities, the total number of education hours (the regular

course hours multiplied by the number of weeks in the semester plus the hours spent in special educational activities), and the number of years employed
3. enter your data
4. print a chart showing employee names, educational activities, and the number of years employed
5. print a graph to examine the relationship between the hours spent in educational activities and the number of years employed.

To begin, log on to your system, and, after receiving the operating system prompt, invoke the SAS System by executing the command

```
SAS
```

Under some operating systems, you are required to identify the type of terminal you are using when you execute the SAS command.[2]

If the cursor is on the command line press the NEXT FIELD key twice to move the cursor to the data entry area. You are now ready to enter the SAS statements and data lines.[3]

Enter the statements and data lines for the DATA step. Remember that SAS statements can begin in any column and that you can split a statement between two lines or enter several statements on one line. You may find your programs easier to read if you follow our convention of beginning DATA, PROC, and global statements in column 1 and indenting other statements. **Screen 14.2** shows the program editor after you have filled in all the lines.

When you have filled in the last line on the program editor, press the FORWARD function key to scroll more lines into view. To control the forward and backward scroll amount, enter

```
VSCROLL n
```

(where *n* is the number of lines) on the command line of the program editor and press the ENTER key. You can also set the vertical scroll amount to HALF or PAGE for scrolling a half or a whole page at a time. HALF is the default vertical scroll amount.

Scroll forward each time you fill in the lines in the program editor screen until you have entered all statements and data lines for the DATA step.

You can scroll back through the example to check for errors. When you are ready to run the DATA step, use the SUBMIT command or function key.

The SAS System displays messages indicating completion of the DATA step in the log screen.[4] Move the cursor to the log screen with the UP or DOWN cursor key or the HOME key, and press the FORWARD or BACKWARD function key to scroll through the log. You can alter the vertical scroll amount by entering the command VSCROLL and the scroll amount on the command line of the log screen, just as you did in the program editor.

Notice that the program editor is now empty so you can enter more statements. To see the statements you entered before, browse the log. To rerun those statements, first return the cursor to the program screen and use the RECALL command or function key; the previously submitted lines are displayed in the program editor. Use the SUBMIT command or function key to submit these program statements to the SAS System for execution again.

Now enter the PROC PRINT and PROC PLOT steps in the program editor. Precede the PROC PRINT statement with the OPTIONS statement specifying the DATE and NUMBER options if you want the date and page number to appear at the top of your output.

```
Command ===>                                                  SAS Log  11:14

Copyright (c) 1985 SAS Institute Inc., Cary, N.C. 27511, U.S.A.
NOTE: XXXXX Version of SAS Release X.XX at SAS Institute Inc.(00000000).
NOTE: LICENSED CPUID MODEL = XXXXXX, SERIAL = 00000000.









--------------------------------------------------------------------------------
Command ===>                                                  Program Editor

00001 data educatn;
00002     input name $ credhrs othered yrsexp;
00003     toted=(credhrs*16)+othered;
00004     cards;
00005 aiken 0 16 3
00006 barker 2 0 3
00007 faulkner 3 8 2
00008 house 3 16 2 _
```

Screen 14.2 Entering Your Program Statements

```
Command ===>                                                  SAS Log  11:14

Copyright (c) 1985 SAS Institute Inc., Cary, N.C. 27511, U.S.A.
NOTE: XXXXX Version of SAS Release X.XX at SAS Institute Inc.(00000000).
NOTE: LICENSED CPUID MODEL = XXXXXX, SERIAL = 00000000.



--------------------------------------------------------------------------------
Command ===>                                                  Program Editor

00009 mailer 0 16 4
00010 noble 0 0 5
00011 parker 2 0 2
00012 radner 3 4 1
00013 restin 3 0 3
00014 rusk 3 4 2
00015 silver 0 8 5
00016 smith 0 0 4
00017 tate 3 8 2
00018 tucker 0 0 4
00019 volker 0 8 5
00020 ;
00021 run;
```

Screen 14.3 Scrolling to Enter More Program Statements

```
Command ===>                                                SAS Log  11:14

NOTE: THE DATA SET WORK.EDUCATN HAS 15 OBSERVATIONS AND 5 VARIABLES.
   20 ;

------------------------------------------------------------------------
Command ===>                                                Program Editor

00001 _
00002
00003
00004
00005
00006
00007
00008
```

Screen 14.4 Messages in the Log Screen

```
Command ===>                                                SAS Log  11:14
  .

NOTE: THE DATA SET WORK.EDUCATN HAS 15 OBSERVATIONS AND 5 VARIABLES.
   20 ;

------------------------------------------------------------------------
Command ===>                                                Program Editor

00001 options date number;
00002 proc print;
00003 run;
00004 proc plot;
00005    plot yrsexp*toted;
00006 run;
00007
00008
```

Screen 14.5 Entering More Program Statements

When you use the SUBMIT command or function key, the SAS System executes your program statements. Because these statements contain PROC steps that produce printed output, the program editor/log screen is removed, and the output screen appears, displaying the results of the PRINT procedure.

Output 14.1 PRINT Procedure Output

```
                                  11:44 FRIDAY, JANUARY 18, 1985   1

OBS     NAME      CREDHRS    OTHERED    YRSEXP    TOTED

  1     AIKEN       0          16         3         16
  2     BARKER      2           0         3         32
  3     FAULKNER    3           8         2         56
  4     HOUSE       3          16         2         64
  5     MAILER      0          16         4         16
  6     NOBLE       0           0         5          0
  7     PARKER      2           0         2         32
  8     RADNER      3           4         1         52
  9     RESTIN      3           0         3         48
 10     RUSK        3           4         2         52
 11     SILVER      0           8         5          8
 12     SMITH       0           0         4          0
 13     TATE        3           8         2         56
 14     TUCKER      0           0         4          0
 15     VOLKER      0           8         5          8
```

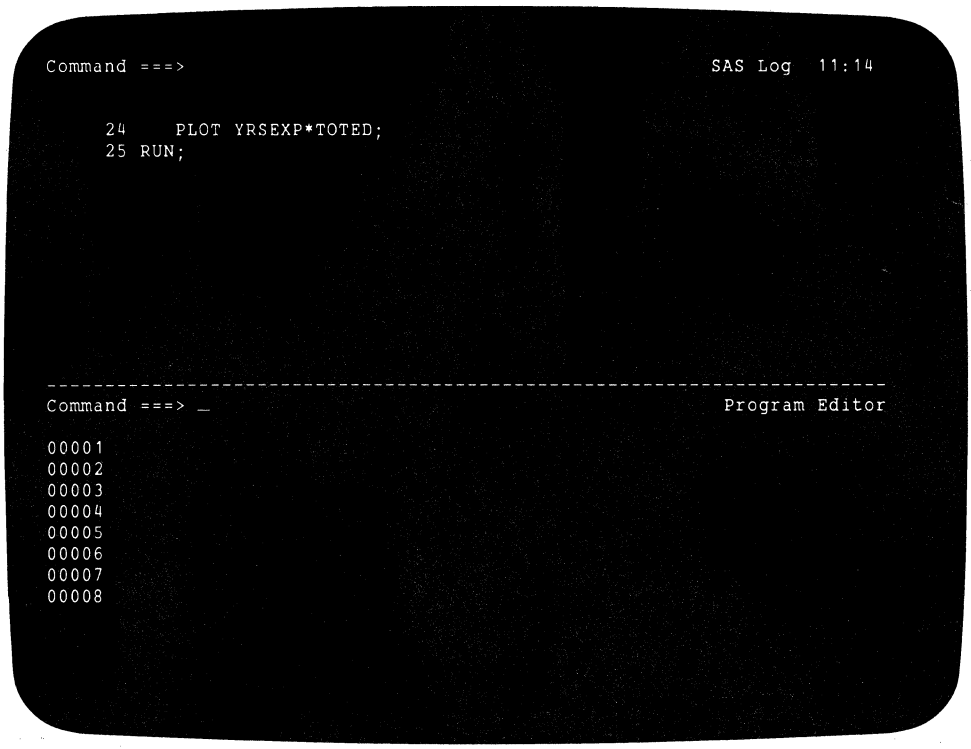
When the PRINT procedure produces only one page of output, as in this example, you will hear a beep from the terminal when that one page has been displayed. Press the END function key to view the output of the PLOT procedure. (By default, function keys number 3 and 15 on the output screen are set to execute the END command.)

Output 14.2 PLOT Procedure Output

```
                                  11:44 FRIDAY, JANUARY 18, 1985 2
          PLOT OF YRSEXP+TOTED  LEGEND: A = 1 OBS, B = 2 OBS, ETC.
RSEXP
   5 +A       B
     |
     |
   4 +B              A
     |
     |
   3 +               A              A              A
     |
     |
   2 +                              A                   A   B      A  A
     |
     |
   1 +                                                  A
     -+-------+-------+-------+-------+-------+-------+-------+-------+
      0       8       16      24      32      40      48      56      64
                                      TOTED
```

You will hear another beep from the terminal, signifying that all output produced by the PLOT procedure has been displayed. You can then browse the output in page-browse mode, enter line-browse mode, or return to the program editor/log screen. Press the COMMAND function key to enter line-browse mode. (By default, function keys number 2 and 14 are set to execute the COMMAND command.) A command line then appears at the top of the output screen. Use the END command or function key to return to the program editor/log screen.

When you return to the program editor/log screen, you see the most recent SAS statements and messages about your job in the log screen. You can view the earlier statements by scrolling backward.



Screen 14.6 SAS Statements and Messages in the Log Screen

You can leave your SAS session in one of several ways. Enter /* in the first two columns of a data line and use the SUBMIT command or function key. Note that the /* must be the last two characters you submit from the program editor; otherwise, it is assumed to be the beginning of a comment. You can also enter

```
ENDSAS;
```

on a data line in the program editor and use the SUBMIT command or function key. On the command line of the program editor or log screen you can type the command

```
BYE
```

and press ENTER to end a SAS session.

NOTES ON USING DISPLAY MANAGER

Active Screen

On the program editor/log screen, only one screen is active at a time. The active screen is defined by display manager as the screen that the cursor is on. When you press ENTER or a function key, only the screen active at that time is affected.

Protected and Unprotected Areas

In display manager screens, protected areas are areas of the display that you are allowed to view but not alter in any way. Unprotected areas are those in which you are allowed to enter and alter text. For example, in the program editor screen, you can type commands in the entry area of the command line, line commands

over the line numbers, and data in the data entry area that follows the line numbers; most of the program editor screen is unprotected. On the other hand, the log and output screens and the help facility are browsing screens. Most of the areas on these screens are protected; you are not allowed to enter or alter any text. The only area on these screens that is unprotected is the entry area of the command line. You can type and edit commands on the command lines of these screens.

The boxed areas below mark the unprotected areas of the program editor/log screen:

Screen 14.7 Unprotected Areas of the Program Editor/Log Screen

Color and Highlighting Attributes

If you are using a color terminal that has extended color and highlighting attributes in an environment that supports them, you can use the following display manager commands to set and change colors and highlighting:

CBANNER	to change the color or highlighting of the border, the line numbers, and screen description.
CPROT	to change the color or highlighting of all protected areas other than the banner. See CBANNER above.
CSOURCE	to change the color or highlighting of SAS source lines in the log screen.
CUNPROT	to change the color or highlighting of unprotected areas.

The colors and highlighting you can use are determined by the features of your terminal. See the *SAS/GRAPH User's Guide* and the *SAS/GRAPH Guide to Hardware Interfaces* for the colors available for your terminal.

Color Use the following abbreviations to specify color attributes in display manager commands:

B blue
R red
P pink
G green
C cyan
Y yellow
W white

Highlighting Use the following abbreviations to specify highlighting attributes in display manager commands:

H highlight
U underline
R reverse video
B blinking

Setting both color and highlighting You can assign an appropriate highlighting attribute at the same time you set a color attribute. Follow the color code with a highlighting code. For example, to set the color attribute of unprotected fields to yellow and the highlighting attribute to underline, use the command

```
CUNPROT Y U
```

To turn off the highlighting attribute, reissue the same color command and specify only a color code. For example, execute

```
CUNPROT Y
```

to allow the color of the unprotected areas of the screen to remain yellow but to remove the underlining.

Function Key Definition Screen

You can display the function key definition screen for the screen you are currently using by executing the KEYS command. You can view the current function key settings, or you can alter a key setting by typing the name of another valid command over the current one. On the function key definition screen you can use the following commands or function keys set to execute these commands:

BACKWARD
CANCEL
END
FORWARD

You can scroll backward and forward to view the entire list of key settings; you can cancel any changes made; you can use the END command, or a function key assigned to execute the END command, to save your altered settings and return to the screen on which you executed the KEYS command.

Note that the function key settings displayed on the screen are those currently active. If you want to execute a command that a function key is not currently set to execute, you must enter the command on the command line. For example, on the program editor and log screens there is no END function key by default because there is no END command for these screens. Execute the END command from the command line to save your settings and exit the function key definition screens of the program editor and log screens.

Entering Multiple Commands

You can enter a series of commands on the command line by separating the commands with semicolons. For example, you can type the BACKWARD MAX scroll command and the FIND command on the command line, separating them with a semicolon, and execute them in order by pressing the ENTER key once.

```
BACKWARD MAX; FIND 'DATA ONE'
```

Function keys and multiple commands You can also combine the use of function keys and the submission of multiple commands. For example, if you set a function key to the command and option BACKWARD MAX followed by a semicolon,

```
BACKWARD MAX;
```

then you can enter

```
FIND 'DATA ONE'
```

on the command line and execute them in order by pressing the key you set to execute BACKWARD MAX; . The procedure executes the BACKWARD MAX command first and then the FIND command.

As a general rule, when you assign commands that do not allow any options to a function key, you may want to follow that command keyword with a semicolon. On the function key definition screen of the program editor screen, for example, you may want to place semicolons after the following commands:

```
BOTTOM;
RECALL;
SPLIT;
TOP;
```

Executing Display Manager Commands Stored in External Files

Executing command list at initialization time You can execute a list of display manager commands stored in an external file automatically at initialization time. When you enter display manager, the system looks for a file, a sequential data set, or a member of a partitioned data set assigned the reserved fileref SASEXEC. If you have assigned SASEXEC to an external file containing a list of display manager commands, those commands are automatically executed. If the system finds no such file, no action is taken, and all default settings are used when you enter display manager.

By executing a series of commands stored in an external file, you can save time and decrease the possibility of errors. Suppose there is a series of commands you want executed each time you enter display manager. You can:

- alter the default active screen
- alter the default colors of protected and unprotected areas of the screen
- execute host-level commands that make SAS data libraries, external files, or data sets available to your session
- copy and submit SAS statements that take you directly into a full-screen procedure.

Using the AUTOEXEC command You can execute a list of display manager commands stored in an external file at any time during your display manager session by executing the AUTOEXEC command. You can follow the AUTOEXEC command with any fileref assigned to a file or data set containing a list of display manager commands. If you do not specify a fileref, the system looks for a file assigned

the fileref SASEXEC by default. Because you can specify a fileref other than SASEXEC, you can use the AUTOEXEC command to execute more than one file of display manager commands during your session. The syntax and definitions of the AUTOEXEC and AUTOSHOW commands are shown in the **Command Glossary** at the end of this chapter.

When you execute the AUTOEXEC command and specify the fileref of a file containing a list of display manager commands, the first command is then executed from the command line of the active screen, which is the screen from which you executed the AUTOEXEC command. You can send commands to all of the display manager screens, one at a time, by including commands that change the active screen. For example, if you issue the AUTOEXEC command from the program editor, the first command is sent to that command line to be executed. Use the LOG command to make the log screen active; all following commands are executed on the log screen. Similarly, you can use the OUTPUT command to make the output screen active and the PROGRAM command for the program editor.

An invalid command If an invalid command is encountered during the execution of your file of commands, an error message is displayed, and the AUTOEXEC processing stops.

Display of executing commands By default these commands are not displayed as they are executed. The command AUTOSHOW ON causes each command to be displayed on the appropriate command line just before execution. AUTOSHOW OFF turns off the display of the AUTOEXEC commands as they are being executed.

The AUTOEXEC and AUTOSHOW commands Enter the AUTOEXEC command to send a list of commands stored in an external file to one or more command lines in display manager. The list of commands is sent to the command line of the screen from which you enter the AUTOEXEC command.

You are not required to specify SASEXEC to execute commands stored in a file associated with the fileref SASEXEC; SASEXEC is the default. If you specify another fileref, the display manager commands stored in the file assigned that fileref are then executed.[5]

Use the AUTOSHOW command with the OFF or ON option to specify whether the commands are to be displayed on the command line as they are executed. AUTOSHOW OFF is the default.

Creating your file of display manager commands The commands you want to be executed from a display manager command line, whether at initialization time or with the AUTOEXEC command, must be stored in an external file, **not** a SAS data set. You can store them in a file, a sequential data set, or a member of a partitioned data set.

Here are some general rules to follow when entering a list of commands to be executed at initialization time or with the AUTOEXEC command:

1. Enter only one command on each line and follow each command with a semicolon. Note: when you are working in display manager and entering commands on the command line directly, one at a time, you do not need to end a command with a semicolon.
2. Do not type beyond the 80th column.[6]
3. You can send commands to each display manager screen one at a time. Use the commands PROGRAM, LOG, OUTPUT, and HELP to designate which screen you want to be active and, therefore, the one on which the following commands will be executed.[7]

4. After you designate the active screen, list only commands that are valid on that screen.
5. The file in which the list of commands is stored **must** be assigned the reserved fileref SASEXEC if you want it to be executed automatically when you enter display manager.[8]

A sample list of stored commands Below is a sample list of commands that you can store in an external file and execute with the AUTOEXEC command or at initialization time:

```
PROGRAM;
INCLUDE MYFILE;
SUBMIT;
CBA G;
LOG;
CURSOR 5;
SPLIT;
CSO G;
VSCROLL PAGE;
OUTPUT;
CPRO B;
PROGRAM;
OUTPUT OFF;
VSCROLL PAGE;
```

These commands are explained in detail below:

PROGRAM	designates that the active screen is the program editor; therefore, all commands that follow this command are sent to the command line of the program editor screen to be executed **until** you specify that another screen is active. In a file executed at initialization time, you are not required to specify PROGRAM because the program editor is the active screen by default.
INCLUDE MYFILE	brings the contents of the external file identified by the fileref MYFILE to the data lines of the program editor screen. This file contains a series of SAS program statements that you want submitted from the program editor to the SAS System. It could, for example, make some SAS data sets available to your session by containing X statements that make fileref assignments through host-level commands; it may also be used to enter an OPTIONS statement.
SUBMIT	submits the program statements that you brought into the data lines of the program editor with the INCLUDE command.
CBA G	sets the color of the banner on the program editor screen to green.
LOG	activates the log screen. All commands that follow the LOG command are sent to the command line of the log screen to be executed **until** you make another screen active.
CURSOR 5	positions the cursor five lines below its default position, the command line of the log screen. This

	command positions the cursor in preparation for the execution of the next command.
SPLIT	tells the screen to split at the cursor location. Because you just positioned the cursor five lines below the command line of the log screen, you have altered the default location of the split between the program editor and log screens.
CSO G	sets the color of the source lines on the log screen. CSO G sets the color of the source lines to green.
VSCROLL PAGE	alters the vertical scroll amount of the log screen and sets it to the full depth of the log screen. HALF is the default.
OUTPUT	activates the output screen in page-browse mode. All commands that follow are now sent to the output screen.
CPRO B	sets the color of the protected text in the output screen. CPRO B sets the color of the protected text to blue.
PROGRAM	activates the program editor screen. All commands that follow are now sent to the command line of the program editor screen.
OUTPUT OFF	indicates that you do not want output produced during your session automatically displayed in the output screen.
VSCROLL PAGE	alters the vertical scroll amount of the program editor screen and sets it to the full depth of the screen. The default is HALF.

This list is just a sample of the kind of commands you may want to store and execute from an external file.

PROGRAM EDITOR SCREEN

Notes on Using the Program Editor Screen

Submitting SAS source lines from the program editor You can think of the program editor as being entirely separate from the SAS System since SAS does not see the lines entered until you are ready to submit your job with the SUBMIT command or function key. After the job has executed, you can resume working in the program editor, entering commands, using editing keys and function keys, and entering and editing program statements or data.

Program editor screen line length When you submit SAS program statements for execution from the program editor screen, the data lines are broken into lengths of 80 columns. It is recommended that you not submit source lines longer than 80 columns. If you do submit longer source lines, do not continue a word from the 80th to the 81st column; it will be divided when submitted.[9]

Executing line commands without line numbers You can execute line commands even if you have set NUMS OFF. You can position the cursor where you want the line command to be executed and then press a function key set to execute a line command. For example, you can position the cursor where you want

a line to split and then press a function key set to execute :TS (the text split line command preceded by a colon). You can also execute a line command from the command line by preceding it with a colon, just as you do when you set a function key. You can type :TS on the command line, position the cursor, and then press the ENTER key.

Line Commands

The line commands available in the program editor screen of display manager are listed below. You can use line commands to move, copy, repeat, insert, and delete lines or blocks of lines; and to designate target locations for moved, copied, and inserted lines. In display manager you can use line commands only on the program editor screen. The log screen, whether you are using it to view the SAS log or SAS HELP, and the output screen are browsing screens; therefore, you cannot use line commands to alter text. Below is a list of line commands. Complete definitions appear at the end of this chapter.

Single commands

A[*n*],B[*n*]
C[*n*]
COLS
D[*n*]
I[*n* | A[*n*] | B[*n*]]
M[*n*]
MASK
O[*n*]
P[*n*]
R[*n*]
TF
TS

Block commands

CC
DD
MM
OO
PP
RR[*n*]

Special shift commands

>[*n*]
<[*n*]
>>[*n*]
<<[*n*]
)[*n*]
([*n*]
))[*n*]
(([*n*]

Command-Line Commands

You can use the command-line commands listed below on the command line of the program editor screen. Definitions of all commands you can use with display manager appear at the end of this chapter.

ASSIGN*
AUTOADD*
AUTOEXEC
AUTOSHOW
BACKWARD
BOTTOM
BYE
CAPS
CBANNER
CHANGE
CPRO(T)
CUNPRO(T)
CURSOR
FIND
FORWARD
HELP
HSCROLL
INCLUDE
KEYS
LEFT
LINESIZE
LOC
LOG
NODMS*
NULLS
NUMBER
OUTPUT
PRINT
PROGRAM
RCHANGE
RECALL
RESET
RFIND
RIGHT
RULE*
SAVE
SCREEN*
SPLIT
SUBMIT
TOP
VSCROLL
X**

LOG SCREEN

Notes on Using the Log Screen

Line length The length of data lines on the log screen is sensitive to the value of certain line size options that you can specify with the OPTIONS statement.[10] You can scroll RIGHT to view data lines longer than the line length of your terminal screen.

---

* not in IBM versions.
** not available under VSE.

Command-Line Commands

You can use the command-line commands listed below on the command line of the log screen. Definitions of all commands you can use with display manager appear at the end of this chapter.

ASSIGN*
AUTOEXEC
AUTOSHOW
BACKWARD
BOTTOM
BYE
CAPS
CBANNER
CPROT
CSOURCE
CUNPROT
CURSOR
FIND
FORWARD
HELP
HSCROLL
KEYS
LEFT
LOC
OUTPUT
PRINT
PROGRAM
RFIND
RIGHT
RULE*
SAVE*
SCREEN*
SPLIT
TOP
VSCROLL
X**

OUTPUT SCREEN

Notes on Using the Output Screen

When the output screen displays output resulting from any procedures you have executed, the procedure displays the first page of output and then waits for your prompt to go to the next page. To display each succeeding page of output, press the ENTER key or any function key **except** the END key.

If you have many pages of output and do not want to wait for the procedure to display them all, one by one, you can press the END key to cause the display manager to skip over all the intervening pages and display only the last page of output. All output is produced whether or not you choose to view each page. The terminal emits a beep when the last page of output is displayed.

When the last page of output is displayed and you receive the audible signal from the terminal, you are in page-browse mode. Now that you have entered page-browse mode, you have three choices; you can:

---

* not in IBM versions.
** not available under VSE.

1. use the function keys operable in page-browse mode to browse the output
2. return to the program editor/log screen by pressing the END key
3. enter line-browse mode by pressing the COMMAND function key. A command line then appears at the top of the screen, and you can either use function keys or enter commands that are operable in line-browse mode.

Browsing more than one set of output If your submitted statements included more than one PROC step that produced output, you cannot return to the program editor/log screen from the last page of the first set of output by pressing the END key. You must reach the last page of the last set of output produced before you can exit the output screen.

When you reach the last page of the first set of output, you can press the COMMAND key to put a command line on the screen and to take you into line-browse mode for browsing that set of output; or you can proceed to the first page of the next set of output by pressing the END key. If you enter line-browse mode, pressing the END key also takes you to the first page of the next set of output. If you do not want to view all pages of a set of output, press the END key again to display the last page of that set of output. At this point, you have the same choices described above. You can browse the output in page-browse mode, browse it in line-browse mode by pressing the COMMAND key, or press the END key to return you to the program editor/log screen.

Using function keys in page-browse mode Because there is no command line available to you in page-browse mode of the output screen, you must use function keys to execute commands. The same function key settings apply to the output screen in both page-browse and line-browse mode. You cannot view the function key definition screen from page-browse mode unless you have a function key for the output screen defined to execute the KEYS command. Use the COMMAND function key to create a command line to take you into line-browse mode where you can execute the KEYS command to view the function key settings for the output screen. (By default function keys number 2 and 14 are set to execute the COMMAND command.) Use the END function key to take you to the next procedure's output or to exit the output screen. (By default function keys number 3 and 15 are set to execute the END command.)

You can also view the function key settings of the output screen by executing the following commands in sequence from the command line of the program editor or the log screen:

```
OUTPUT; KEYS
```

The OUTPUT command takes you to the OUTPUT screen; the KEYS command then displays the function key definition screen of the output screen.

Linesize on output screen The maximum line length depends on the operating system under which you are using the SAS System.[11]

Command-Line Commands in Line-Browse Mode

When viewing the output screen, you must enter line-browse mode to be able to enter commands directly. After hearing a beep from the terminal, which signifies that all of the output has been displayed and that you have entered page-browse mode, press the COMMAND function key to create a command line on the output screen and to enter line-browse mode. Then you can use the command-line commands listed below. Definitions of all commands you can use with display manager appear at the end of this chapter.

ASSIGN*
AUTOEXEC
AUTOSHOW
BACKWARD
BOTTOM
BYE
CAPS
CBANNER
COMMAND
CPROT
CUNPROT
CURSOR
END
FIND
FORWARD
HELP
HSCROLL
KEYS
LEFT
LOC
LOG
OUTPUT
PRINT
PROGRAM
RFIND
RIGHT
RULE*
SAVE*
SCREEN*
SPLIT*
TOP
VSCROLL
X**

COMMAND GLOSSARY

Line Commands

A[*n*],**B**[*n*] A line command that marks the target position of a *source* line(s), a line or lines being moved or copied with a C, M, P, CC, MM, PP, or INCLUDE command. Indicate an A (after) on the line number of the line you want the source line(s) to follow. Use a B (before) to mark the line you want the source line(s) to precede. After the A or B, you can specify the number of times you want the source line or lines to be duplicated. Follow A or B with some number *n* and a blank space.

CMS, OS, VM/PC, and VSE: You cannot use the target commands with the INCLUDE command.

---

* not in IBM versions.
** not available under VSE.

C[*n*] A line command that copies one or more lines to another location in the file, indicated by a target line command. Indicate C on the line number of the line to be copied. Then indicate an A on the number of the line you want the copied line to follow, a B on the number of the line you want it to precede, or an O or OO on the line or lines you want it to overlay. You can also specify *n* number of lines to be copied. Follow C with some number *n* and a blank space.

CC A line command that copies a block of lines to another location in the file, indicated by a target line command. Indicate CC on the line numbers of the first and last lines of the block of lines to be copied.

COLS A line command that creates a special line indicating the column numbers across your display screen. The column indicator line appears above the line on which you execute the COLS command. The COLS line is a special line that is not submitted when you submit program statements from the program editor. Use the RESET key or command or delete the line to remove the COLS line.

D[*n*] A line command that deletes one or more lines. Indicate D on the line number of the line to be deleted. By default, one line is deleted. To delete more than one line, follow D with some number *n* and a blank space.

DD A line command that deletes a block of lines. Indicate DD on the line numbers of the block of lines to be deleted.

I[*n*] This line command is used to insert one or more new lines. By default one line is inserted after the line on which you execute the I command. To insert more than one line, follow I with some number *n* and a blank space. See also the IA and IB commands below.

See the MASK command to insert lines with a defined content other than a blank line. See the TS (text split) command to insert space within a line of text.

IA[*n*] Use this line command to insert one or more lines after the line on which you enter the IA (insert after) command. By default only one line is inserted. See also the I and IB commands.

IB[*n*] Use this line command to insert one or more lines before the line on which you enter the IB (insert before) command. By default only one line is inserted. See also the I and IA commands.

M[*n*] A line command that moves one or more lines to another location in the file, indicated by a target line command. Indicate M on the line number of the line to be moved. If you move more than one line, follow M with some number *n* and a blank space. Then specify an A on the line number of the line you want

the moved line to follow, a B on the line you want it to precede, or an O on the line you want the moved line or lines to overlay.

MASK A line command that defines the initial contents of a new line. Type MASK on any line number and press ENTER. Then type on that line whatever characters you want to be repeated and press ENTER again. After you define a MASK, a line with the contents of the MASK line is inserted when you use the I (insert) line command.

The MASK remains in effect throughout the session. To redefine it, simply repeat the steps described above. To return to the default, a blank line, use the same process and leave the line blank.

MM A line command that moves a block of lines to another location in the file, indicated by a target line command. Indicate the block to be moved with an MM line command on the first and last line numbers of the block.

>[*n*] A line command that shifts data one or more spaces to the right. Indicate > or > followed by some number *n* and a blank space on the line number of the line to be shifted. Note that a **data** shift command allows no loss of data. The default is one space.

<[*n*] A line command that shifts data one or more spaces to the left. Indicate < or < followed by some number *n* and a blank space on the line number of the line to be shifted. Note that a **data** shift command allows no loss of data. The default is one space.

> >[*n*] A line command that shifts a block of lines one or more spaces to the right. Indicate >> or >> followed by some number *n* and a blank space on the line number of the first line of the block to be shifted and another >> or >>*n* on the last line number of the block. Note that a **data** shift command allows no loss of data. The default is one space.

< <[*n*] A line command that shifts a block of lines one or more spaces to the left. Indicate << or << followed by some number *n* and a blank space on the line number of the first line of the block to be shifted and another << or <<*n* on the last line number of the block. Note that a **data** shift command allows no loss of data. The default is one space.

)[*n*] A line command that shifts columns of data one or more columns to the right. Indicate) or) followed by some number *n* and a blank space on the line number of the line to be shifted. Note that a **column** shift command, unlike a **data** shift command, can cause loss of data. The default is one column.

([*n*] A line command that shifts columns of data one or more columns to the left. Indicate (or (followed by some number *n* and a blank space on the line number

of the line to be shifted. Note that a **column** shift command, unlike a **data** shift command, can cause loss of data. The default is one column.

))[*n*] The column shift line command shifts a block of lines one or more columns to the right. Indicate)) or)) followed by some number *n* and a blank space on the line number of the first line of the block to be shifted and another)) or))*n* on the last line number of the block. Note that a **column** shift command, unlike a **data** shift command, can cause loss of data. The default is one column.

(([*n*] A line command that shifts a block of lines one or more columns to the left. Indicate ((or ((followed by some number *n* and a blank space on the line number of the first line of the block to be shifted and another ((or ((*n* on the last line number of the block. Note that a **column** shift command, unlike a **data** shift command, can cause loss of data. The default is one space.

O[*n*] A line command that marks the target position of a C, M, P, CC, MM, or PP line command. Indicate the O (overlay) line command on the line number of the line you want the contents of the *source line*, the moved or copied line, to overlay. Characters from the source line overlay blank or null spaces on the *target line*, the line marked with the O line command. After the O line command you can specify the number of target lines you want the source line or lines to overlay. Follow O with some number *n* and a blank space.

If any characters occupy the same positions on the source and target lines, the characters on the target line remain, and characters from the source line do not appear. If you are executing the M (move) command, you receive an error message, and the line intended to be moved remains in its original position.

OO A line command that marks a block of lines, identified by an OO on the line numbers of the first and last lines of the block, as the target position of source lines, lines marked with the C, M, P, CC, MM, or PP line commands. See the O line command.

P[*n*] A line command that copies or overlays one or more lines to another location in the file, indicated by a target line command. The P (pattern) line command is similiar to the C (copy) line command; the difference is that the system remembers the line marked with the P line command and copies it to the new location each time you enter a target line command until you remove the P line command. You can remove it with the RESET key or command or by using any of the editing keys you normally use to erase or remove characters, such as the EOF key, the character delete key, or the space bar. Indicate P on the line number of the source line, the line you want to copy. Then indicate an A on the number of the line you want the source line to

follow, a B on the number of the line you want it to precede, or an O or OO on the line or lines you want it to overlay. You can also specify *n* number of lines to be copied. Follow P with some number *n* and a blank space.

PP A line command that copies a block of lines to another location in the file, indicated by a target line command. The PP (pattern block) command is similiar to the CC line command; the difference is that the system remembers the block of lines marked with the PP line command and copies it to the new location each time you enter a target line command until you remove the PP line command. You can remove it with the RESET key or command or by using any of the editing keys you normally use to erase or remove characters, such as the EOF key, the character delete key, or the space bar. Indicate PP on the first and last line numbers of the block of lines you want to designate as the source lines, the lines you want to copy. Then indicate an A on the number of the line you want the source lines to follow, a B on the number of the line you want them to precede, an O or OO on the first and last lines of the block you want the source lines to overlay.

R[*n*] A line command that repeats a line one or more times immediately following. Indicate R on the line number of the line to be repeated. To repeat a line more than once, follow R with some number *n* and a blank space.

RR[*n*] A line command that repeats a block of lines immediately following that block. Indicate RR on the line numbers of the first and last lines of the block of lines to be repeated. You can also specify how many times you want the block of lines repeated; follow RR with some number *n* and a blank space. You can specify *n* on the first or last RR command or on both.

TF A line command that flows a paragraph or an indicated block of text by removing trailing blanks from each line. You can use the TF (text flow) line command to move text into wasted space left at ends of lines, especially after performing insertions and deletions.

TS A line command that inserts a blank line for inserting new text. Either type TS (text split) on a line number, position the cursor where you want the text to split, and press ENTER, or position the cursor and press the :TS function key. See the I (insert) command for inserting space between lines rather than within the text of a line.

Command-Line Commands

Command-line commands that are available in display manager screens are defined below:

ASSIGN filename fileref

AOS/VS, PRIMOS and VMS only: Use the ASSIGN command to assign a fileref to an external file. You can specify a fully qualified file

name, according to the requirements of the operating system you are using, or just an individual file name (and type under VMS) if it is in a directory to which you currently have access.

For example, to assign the fileref PROG1 to an AOS/VS, PRIMOS, or VMS fully-qualified file name, you can execute the ASSIGN command in the form

```
ASSIGN :UDD:youruserdir:PROG.SAS PROG1

ASSIGN [yourdir]PROG.SAS PROG1

ASSIGN <masterfiledirectory>userfiledirectory>PROG.SAS PROG1
```

See the INCLUDE and SAVE commands.

CMS, OS, and VM/PC: You can execute host-level commands before entering display manager or after entering display manager with the X command or statement.

VSE: You can place a /FILE statement, an ICCF control language statement, in the ICCF procedure that invokes the SAS System. You cannot use the X command or statement in the ICCF environment. See the INCLUDE and SAVE commands.

AUTOADD [ON | OFF]

AOS/VS, PRIMOS, and VMS only: The AUTOADD ON command adds data entry lines to the bottom of the screen each time you scroll forward so that you do not have to use an INSERT command to insert new lines for entering new data or text. If line numbers are on (NUMS ON), these lines are numbered each time you scroll foward. If you execute AUTOADD OFF, the editor does not add new lines for data entry each time you scroll foward.

AUTOEXEC [SASEXEC | *fileref*]

Enter the AUTOEXEC command to send a list of commands stored in an external file to one or more command lines in display manager. The first command in the list is sent to the command line of the screen from which you enter the AUTOEXEC command.

If you do not specify a fileref that you assigned to an external file through a host-level command, the procedure looks for a file assigned the fileref SASEXEC by default. If you specify another fileref, the display manager commands stored in file associated with that fileref are then executed.

AUTOSHOW OFF | ON

Use the AUTOSHOW command with the OFF or ON option to specify whether commands being executed from an external file are to be displayed on the command line as they are executed. AUTOSHOW OFF is the default.

BACKWARD [*n* | MAX]

In addition to using a function key, you can scroll toward the top of the screen with the BAC command. The amount of scroll is controlled by the VSCROLL command.

You can specify a particular number of lines to scroll backward by entering BAC *n* (*n* being the number of lines you want to scroll backward). You can scroll the maximum amount by entering BAC MAX or BAC M. The scroll value specified with the BAC command, either *n* or MAX, is operative only for that scroll; it temporarily overrides but does not alter the default scroll value or the one set by the VSCROLL command.

BOTTOM

Use the BOTTOM or BOT command to scroll the last line of text to the bottom of the screen.

BYE

You can use the BYE command to end a SAS session. You can execute the BYE command from the command line of the log or program editor screen.

CAPS OFF | ON

When you execute CAPS ON, all text entered, as well as text on lines that have been modified, is translated into uppercase letters when you press ENTER or a function key. All text is left as entered when CAPS OFF is in effect. The CAPS command affects only the screen on which it is entered and is in effect for the remainder of the SAS session or until changed by another CAPS command. The default is CAPS ON in minicomputer versions and CAPS OFF in IBM versions.

CBANNER *color* [*highlight*]

The CBA command changes the color or color and highlighting attributes of the banner lines. These lines are the borders, line numbers if any, and command line. Specifying CBA affects only the screen on which the command is entered and is in effect for the remainder of the SAS session or until changed by another CBA command.

CHANGE *string1 string2* [**NEXT | FIRST | LAST | PREV | ALL**] [**WORD | SUFFIX | PREFIX**]

Use the CHANGE command to change one or more occurrences of *string1* to *string2*. Follow the CHANGE command with string of characters to be changed, a space, and then the new string, or type the strings on the command line and press the CHANGE function key.

You can specify on the CHANGE command that the system search for and alter the NEXT occurrence of the specified string after the current cursor location; the FIRST occurrence of the string in the file, regardless of your current cursor location; the LAST occurrence of the string in the file; or the PREVious occurrence. If you specify ALL, you receive a message that reports how many times the string occurs in the entire file, and each occurrence is changed. By default, the CHANGE command searches for and changes the NEXT occurrence of the specified string after the current cursor location.

You can also specify one of the following options: PREFIX, SUFFIX, or WORD. If you do not specify one of these, *string1* is changed to *string2*, regardless of context.

In the CHANGE command, as in the FIND command, a WORD is one or more symbols preceded and followed by a delimiter. A delimiter is any symbol other than an uppercase letter, a lowercase letter, a digit, or an underscore.

Remember to use single quotes to enclose strings with special characters or embedded blanks. Single word strings require no quotation marks. For example,

```
C YOUR MY
C 'YOUR DATA SET' 'MY DATA SET'
```

Also enclose your string in single quotes if CAPS are ON and you do not want lowercase letters in the string translated into uppercase letters.

If your string contains a single quotation mark, such as

```
C "Bob's" "Bill's"
```

enclose it in double quotation marks.

You can combine the use of the CHANGE command and RFIND function key (or command). For example, after you enter a CHANGE command, you can press the RFIND function key to locate the next occurrence of *string1* before pressing RCHANGE to change it to *string2*.

Also see the RCHANGE, FIND, and RFIND commands.

COMMAND

Use the CMD command, assigned to a function key, in page-browse mode of the procedure output screen to put a command line at the top of the screen, taking you into line-browse mode. Execute the command again to remove the command line and return you to page-browse mode.

CPROT *color* [*highlight*]

The CPROT command changes the color or color and highlighting attributes of protected fields on the log or output screen to those specified for the remainder of the SAS session or until changed by another CPROT command. This color or color and highlighting attributes become the default for the protected fields only for the screen on which you enter the command.

The CPROT command has no effect on the program editor screen since its protected areas are governed by the CBANNER command.

CSOURCE *color* [*highlight*]

The CSO command changes the color or color and highlighting attributes of SAS source line on the log screen to those specified. The CSO command is in effect for the remainder of the SAS session or until changed by another CSO command.

CUNPROT *color* [*highlight*]

The CUN command changes the color or color and highlighting attributes of unprotected fields to those specified for the remainder of the SAS session or until changed by another CUN command. This color or color and highlighting attributes become the default for all unprotected fields that are not otherwise defined by color or highlighting attribute commands. Note: the CUNPROT command affects the unprotected fields only for the screen on which it is executed.

CURSOR [*rownumber*] | [*rownumber colnumber*]

The CURSOR command, without any options specified, is designed to be executed with a function key. Press the CURSOR key to return the cursor to the command line of whatever screen you are currently using.

Specifying options with the CURSOR command is especially useful when executing a list of commands stored in an external file. (See the AUTOEXEC command.) You can use the CURSOR command to position the cursor in preparation for the execution of a following command. For example, if you execute

```
CURSOR 10
```

on the command line of the log screen and then execute the SPLIT command, you can alter the location of the split between the

program editor and the log screens. Note: do not move the cursor outside the boundaries of the screen from which you want the next command to be executed.

END

In line-browse mode of the procedure output screen, the END command either returns you to the program editor/log screen or takes you to the first page of the next procedure's output. In page-browse mode, it takes you to the next page of output or, if you are already viewing the last page, returns you to the program editor/log screen.

FIND *characterstring* [**NEXT | FIRST | LAST | PREV | ALL**] [**PREFIX | SUFFIX | WORD**]

Use the FIND command to search for a specified string of characters. Remember to enclose the string in single quotes if it contains embedded blanks or special characters. You can execute the FIND command by entering it directly or by typing the character string on the command line and pressing the FIND function key.

You can specify in the FIND command that the system search for the NEXT occurrence of the specified string after the current cursor location; the FIRST occurrence of the string in the file, regardless of your current cursor location; the LAST occurrence of the string in the file; or the PREVious occurrence. If you specify ALL, you receive a message that reports how many times the string occurs in the entire file. By default, the FIND command searches for the NEXT occurrence of the specified string after the current cursor location.

You can also specify one of the following options: PREFIX, SUFFIX, or WORD. If you do not specify one of these options, the system searches for each occurrence of the string, regardless of context.

In the FIND command, a WORD is one or more symbols preceded and followed by a delimiter. A delimiter is any symbol other than an uppercase letter, a lowercase letter, a digit, or an underscore. For example,

```
ABC123
```

is a WORD, but

```
ABC$123
```

is two WORDs separated by the delimiter $. A PREFIX or SUFFIX is treated just as its grammatical definition. In the first example, ABC123, you could specify ABC as a PREFIX and 123 as a SUFFIX. In the second example, ABC$123, ABC and 123 are WORDs, not a PREFIX and a SUFFIX.

Remember to use single quotes to enclose strings with special characters or embedded blanks. For example,

```
F YOUR
F 'YOUR DATA SET'
```

Also enclose your string in single quotes if CAPS are ON and you do not want lowercase letters in the string translated into uppercase letters.

If your string contains a single quotation mark, such as

```
F "Bob's"
```

enclose it in double quotation marks.

See also the CHANGE, RCHANGE, and RFIND commands.

FORWARD [*n* | **MAX**]

In addition to using a function key, you can scroll toward the bottom of the screen with the FOR *n* command. The amount of scroll is controlled by the VSCROLL command.

You can specify a particular number of lines to scroll forward by entering FOR *n* (*n* being the number of lines you want to scroll foward). You can scroll the maximum amount by entering FOR MAX or FOR M. The scroll value specified with the FOR command, either *n* or MAX, is operative only for that scroll; it temporarily overrides but does not alter the default scroll value or the one set by the VSCROLL command.

HELP [*topic*]

When you enter the HELP command, SAS HELP, an on-line help facility, appears on your display screen. You can specify a topic at the same time that you execute the HELP command. If you do not specify a topic, a list of allowed topics is displayed.

CMS, OS, VM/PC, and VSE: You can execute HELP from any of the three display manager screens. SAS HELP is then displayed, occupying the entire display area. Use the END command or function key to return display manager to the screen.

AOS/VS, PRIMOS, and VMS: You can execute HELP from any of the three display manager command lines, but SAS HELP is always displayed in the upper portion of the program editor/log screen. You can browse help information by using the same commands and function keys that are available in the log screen. If you execute HELP from the output screen, you must return to the program editor/log screen to view the help information. After viewing the help information, you can return the SAS log to the log screen by entering the LOG command or submitting at least one statement from the program editor.

HSCROLL HALF | PAGE | *n*

The HSCROLL command sets the horizontal scroll amount, the amount the active screen scrolls when you press the LEFT or RIGHT scrolling keys. If you specify PAGE, the scroll amount is equal to the entire width of the display area. If you specify HALF, the scroll amount is one-half of the display area. HALF is the default.

INCLUDE *fileref* | *fileref(membername)*
INCLUDE *fileref* | *'filename'*
INCLUDE *singlelinenumber* | *firstline lastline*

The INCLUDE command allows either an entire external file or lines from the log screen to be displayed in the program editor screen. When including lines from the log screen, you can specify particular lines. The included lines are inserted at the bottom of the list of SAS source statement lines or wherever you specify with the A, B, or O line command. When specifying lines on the log screen, you can optionally use a hyphen or a colon between your first and last line specifications.

CMS, OS, and VM/PC: If you are working with an external file, you should use a host-operating system command to assign it a fileref. You can execute it before invoking SAS or from display manager with an X command.

AOS/VS and VMS: You can bring an external file to the program editor screen with the INCLUDE command in one of four ways. The *filename* can be a fully qualified filename, according to the

conventions of the operating system you are using, or the name (and type under VMS) of a file in a directory to which you currently have access.

1. Assign a fileref to a file name with a host-level command either before invoking SAS or from display manager with an X command.
2. Assign a fileref to a file name with the display manager ASSIGN command.
3. Specify an actual file name by enclosing it in single quote. In AOS/VS, the file name specified must be in your current working directory; in VMS, it must be specified in your home directory. Under VMS, you must include the file type.
4. Specify an unassigned fileref; the system then assigns that specified fileref to an existing file of the name *fileref*.SAS in your current working directory.

PRIMOS: See the note above for AOS/VS and VMS. You can use all but the first method listed.

VSE: You can place a /FILE statement, an ICCF control language statement, in the ICCF procedure that invokes the SAS System. You cannot use the X command or statement in the ICCF environment. If you are including a member of an ICCF library, the fileref must be ICCF.

KEYS

Use the KEYS command to display the function key definition screen. You can view the current settings or alter them and store your new ones. You can scroll forward and backward on the function key definition screen; use the END command or function key to return to the previous screen.

LEFT [*n* | **MAX**]

This command scrolls the screen *n* spaces to the left. By default the screen scrolls half of its width. To override the scroll amount temporarily, enter LEFT and the number of spaces you want the screen to scroll. Enter LEFT MAX to scroll the screen to the left boundary.

LINESIZE *n*

Use the LINESIZE command to alter the line length of data lines on the program editor screen; *n* can be any number between the width of your screen minus 1 and 256. The LINESIZE command allows you to use the INCLUDE command to display files of long data lines in the program editor for editing, for example, saved SAS output files. Note: if you specify a LINESIZE shorter than your current line length, your data lines will be truncated.

AOS/VS, PRIMOS, and VMS: The maximum line length of the program editor is 256 columns.

CMS, OS, VM/PC, and VSE: The maximum line length of the program editor is 80 columns.

LOC *n*

Use the LOC or LOCATE command, followed by some line number *n*, to scroll that line to the top of the display screen.

LOG

The LOG command moves the cursor to the command line of the log screen and makes it the active screen.

AOS/VS, PRIMOS, and VMS only: You can also use the LOG command to return the SAS log to the upper portion of the program editor/log screen, replacing the output screen or help information.

NODMS

AOS/VS, PRIMOS, and VMS only: The NODMS command takes you from display manager to line mode and turns off the SOURCE option. The screen is cleared, and you receive the SAS prompt. The SAS System is then ready to receive your next SAS statement. Leaving display manager does not close the file that is currently open or end your SAS session. When you leave display manager and enter line mode, all of your current WORK data sets are still available to you.

NULLS ON | OFF

When you execute NULLS ON, all data lines are padded with nulls instead of blanks. Turning NULLS ON allows you to use the INSERT editing key to insert characters between text already entered on a line.

If you execute NULLS OFF, the data lines are padded with blanks. To use the INSERT editing key, you must first use the EOF editing key to turn those blanks into nulls.

The effect of the NULLS command on how a file is saved with the SAVE command depends on the host operating system you are using.

AOS/VS, PRIMOS, and VMS: If NULLS are ON when you save the file, the file is saved with no trailing blanks. If NULLS are OFF, trailing blanks are saved.

CMS, OS, VM/PC, and VSE: If you are saving a file in a sequential or partitioned data set with a fixed block record format, the setting of the NULLS command has no effect. If you are saving to a data set with a variable block record format, NULLS ON causes no trailing blanks to be saved, and NULLS OFF causes the trailing blanks to be saved.

NUMBERS ON | OFF
NUMS ON | OFF

Use the NUMS ON command to create line numbers for data lines on your screen. If your screen already contains text, all data are shifted to the right, and the line numbers appear on the left. Execute NUMS OFF to remove line numbers and shift text back to the left. NUMS ON is the default on the display manager program editor screen.

If your data lines are numbered, you can type line commands directly over line numbers. If you set NUMS OFF, you can still execute line commands by entering them from the command line or by using a function key. Precede the line command with a colon when executing it on the command line rather that on a line number. For example, type :TS (text split) or :I (insert) on the command line, position the cursor, and then press ENTER. When using a function key, position the cursor, and then press the function key set to execute :TS.

OUTPUT
OUTPUT [ON | OFF]
OUTPUT [TOP]

The OUTPUT command, executed with no option specified, replaces the log screen with the output screen to allow you to view output while keying more SAS statements in the program editor screen.

The OUTPUT command followed by ON or OFF specifies whether procedure output is displayed in the output screen immediately while the procedure is executing (ON) or held for later viewing (OFF). If OUTPUT OFF is in effect, specify OUTPUT to view your output. The default is OUTPUT ON.

AOS/VS, PRIMOS, and VMS only: The OUTPUT TOP command replaces the log screen in the upper portion of the program editor/log screen with the output screen.

PRINT [PROGRAM | RECALL | LOG | OUTPUT]
[fileref | fileref(membername)]

CMS, OS, VM/PC and **VSE only:** You can use the PRINT command to print the entire information stream of a display manager screen, not just what is currently displayed.

You can print the stream of information in the program editor by specifying the PROGRAM option; print all statements previously submitted from the program editor with the RECALL option; the stream of information in the log screen with the LOG option; and all procedure output with the OUTPUT option.

The contents of the log, program editor or output screen, or all previously submitted statements are sent to the default system printer.

You can also send the contents to an external file by specifying its previously assigned fileref or fileref followed by a member name.

PRINT [ALL]

AOS/VS, PRIMOS, and VMS only: With the PRINT command, you can print what is currently showing in the screen on the default system printer. By specifying ALL, you can print the entire contents of the current screen, not just what is visible at the time you enter the PRINT command.

PROGRAM

The PROGRAM command moves the cursor to the command line of the program editor screen and makes it the active screen.

RCHANGE

Use the RCHANGE or RC command to continue to FIND and CHANGE a string of characters previously specified in a FIND or CHANGE command.

See also the CHANGE, FIND, and RFIND commands.

RECALL

Use the RECALL command to bring back to the program editor screen any lines you have submitted to SAS since you entered display manager. The system adds the recalled lines to the top of the program editor screen in front of any other nonblank lines already entered. After you have used the RECALL command to recall the most recently submitted block of statements, you can enter it again to recall the next most recent block.

RESET

You can use the RESET command on the program editor screen to remove any pending line commands, any conflicting line commands, and the COLS and MASK lines.

RFIND

Use the RFIND or RF command to continue the search for a string of characters previously specified in a FIND or CHANGE command.

See also the FIND, CHANGE, and RCHANGE commands.

RIGHT [*n* | **MAX**]
This command scrolls the screen *n* spaces to the right. By default the screen scrolls half of its width. To override the scroll amount temporarily, enter RIGHT and the number of spaces you want the screen to scroll. Enter RIGHT MAX to scroll the screen to the right boundary.

RULE OFF | ON
AOS/VS, PRIMOS, and VMS only: You can use the RULE ON command to display a ruler on the message line beneath the command line. The ruler marks vertical columns and moves with the data lines as you scroll right and left.

The ruler is temporarily overridden by any message that SAS displays on the message line, but the ruler returns when the message is removed. The default is RULE OFF.

SAVE *fileref* | *fileref(membername)*
CMS, OS, VM/PC and **VSE:** The SAVE command writes all program lines in the program editor screen into a file that you designate with a fileref (DDname).

See also the NULLS command.

SAVE *fileref* | *'filename'*
AOS/VS, PRIMOS, and VMS: The SAVE command writes the entire information stream of the display manager screen from which you execute the SAVE command into a file that you designate with a fileref or an actual file name. You can use a host operating system command to assign a fileref to a sequential file **before** you enter display manager (except under PRIMOS), or you can use the SAS display manager command ASSIGN once you have entered display manager. If you specify an unassigned fileref, the system writes the information stream into a file named *fileref*.SAS in your current working directory. If you specify an actual file name, enclose it in single quotes. The *filename* can be either a fully qualified file name, according to the conventions of the operating system you are using, or just the name (and type under VMS) of a file in a directory to which you currently have access.

See also the NULLS command.

SCREEN OFF | ON
AOS/VS, PRIMOS, and VMS only: You can use the SCREEN command to store the contents of the display manager screen in an external file, **not** a SAS data set or catalog. After you have entered the SCREEN ON command, a file is immediately created with the exact contents of what is displayed on the physical screen. Each time you press ENTER, SUBMIT, or any function key thereafter, another file is created. Each external file containing the contents of a screen is given a file name in the form

```
SCRNnnnn.SCR
```

The first file created is given the name SCRN0001, and each file is given the extension SCR. The screen file is stored in your current working directory.

Each time you begin a SAS session, the system begins numbering screen files with SCRN0001.SCR. If SCRN0001.SCR already exists, the newly created one is written over the existing one.

Enter the SCREEN OFF command when you no longer want the contents of the screen copied to external data files. The default is SCREEN OFF.

SPLIT

You can indicate where you want the screen to split between the log and program editor screens with the SPLIT command. Enter SPLIT on the command line, position the cursor where you want the split to occur, and press ENTER; or position the cursor and press the SPLIT function key. Note: you **must** position the cursor within the boundaries of the screen on which you entered the SPLIT command. Moving the cursor into the area of another screen activates the command line of that screen.

SUBMIT ['*SAS statement;*']

Use the SUBMIT key or command with no option specified to submit SAS program statements from the data lines of the program editor screen

Optionally, you can submit a SAS program statement directly from the command line with the SUBMIT command. You can type the command, one or more SAS statements, each followed by a semicolon, on the command line and press ENTER, or you can type just the statements with semicolons on the command line and press the SUBMIT function key.

TOP

Use the TOP command to scroll the first line of text to the top of the screen.

VSCROLL HALF | PAGE | *n*

The VSCROLL command sets the vertical scroll amount, the amount the active screen scrolls when you press the FORWARD or BACKWARD scrolling function keys. If you specify PAGE, the scroll amount is the number of data lines currently displayed. If you specify HALF, the scroll amount is one-half of the display area. If you specify *n*, the scroll amount is *n* lines. HALF is the default.

X *hostcommand*
X '*hostcommand*'

You can issue many host commands by entering an X on the command line followed by one or more host commands. Note that in minicomputer environments, the host command or string of commands **must** be enclosed in single quotes if it contains special characters; in IBM environments you are not allowed to enclose the host command in single quotes.

See the documentation available for SAS under the operating system you are using for any restrictions on the execution of host operating system commands.

VSE: The X command is not currently supported.

NOTES

1. If you are using one of the terminals listed below in **Table 14.1**, use the following charts to determine the keys that have been assigned special editing functions in SAS full-screen procedures and display manager. See **Figure 14.1** for a description of the functions of these editing keys. If you are using a terminal other than one of those listed below that does not have keys matching the words

or symbols shown in **Figure 14.1**, consult your SAS representative. If you are using an IBM terminal in the 3270-series, you can disregard this section.

Table 14.1 Full-Screen Editing Functions Assigned to Terminal Keys

Display Manager Editing Key	PT45 Key	PST100 Key	PT25 Key
â (INSERT)	ichar or shift/ del	insert or backspace	shift/del
∂ (DELETE)	dchar or backspace	delete or del	backspace
ERASE EOF	line feed	erase	shift/f8
HOME	home	home	home
REFRESH	control/ clear	pf10	control/erase
NEW LINE	return or enter	return or enter	new line or enter
NEXT FIELD	f13	f2	shift/f5
PRIOR FIELD	f14	f3	shift/f6
ENTER	f3 or f15	f1 or pf13	f3 or shift/f7
MOVE CURSOR UP	↑	↑	↑
MOVE CURSOR DOWN	↓	↓	(not available)
MOVE CURSOR LEFT	←	←	←
MOVE CURSOR RIGHT	→	→	→

Continued on next page

Table 14.1 *Continued*

Display Manager Editing Key	TEK4105 Key	VT100 Key	DASHER Key
â (INSERT)	backspace	backspace	C1
(DELETE)	rub out	delete	DEL
ERASE EOF	line feed	line feed	erase EOL
HOME	- (minus sign on numeric keypad)	- (minus sign on numeric keypad)	home
REFRESH	, (on numeric keypad)	, (on numeric keypad)	erase page
NEW LINE	return	return	new line or enter
NEXT FIELD	0 (on numeric keypad)	0 (on numeric keypad)	C4
PRIOR FIELD	. (on numeric keypad)	. (on numeric keypad)	C3
ENTER	enter	enter	CR
MOVE CURSOR UP	F1	↑	↑
MOVE CURSOR DOWN	F2	↓	↓
MOVE CURSOR LEFT	F3	←	←
MOVE CURSOR RIGHT	F4	→	→

2. If you are using one of the terminals listed in **Table 14.1**, you must identify it to the SAS System. If you have already invoked the SAS System, you can use the statement

```
OPTIONS DMS;
```

and then receive a prompt, asking you to identify the devicename. When invoking the SAS System, you can specify the devicename with the FSDEVICE= (FSD=) option. For example

AOS/VS	SAS/FSD=D1
PRIMOS	SAS -FSD=PT25
VMS	SAS/FSD=VT100

Table 14.2 Devices to Be Identified with the FSDEVICE= Option

TERMINALS: Manufacturer	Model No.	Devicename
Data General	6052	DG6052 D1 (alias)
	6053	DG6053 D2 (alias)
	6093	DG6093 D3 (alias)
	D100	D100
	D200	D200
	D210	D210
	D211	D211
	D400	D400
	D410	D410
	D450	D450
	D460	D460
	D470C	D470C
	G300	G300
	G500	G500
Note: devicename DG605X is an alias for all Data General terminals.		
Digital Equipment	VT52	VT52
	VT100	VT100
	VT125	VT125
	VT131	VT131
	VT132	VT132
	VT220	VT220
	VT240	VT240
	VT241	VT241
Prime Computer (PRIME)	PT25	PT25
	PT45	PT45
	PST100	PST100
Tektronix	4105	TEK4105

3. **AOS/VS, PRIMOS, and VMS:** By default the program editor translates all characters into uppercase. If you want to prevent the translation, execute the

CAPS OFF command on the program editor screen either from the command line or with a function key before entering program statements.

CMS, OS, VM/PC, and VSE: By default the program editor does not translate all characters into uppercase. Execute the CAPS ON command if you want text entered or lines altered after you execute the command to be translated into uppercase.

4. **AOS/VS, PRIMOS, and VMS:** A message indicating the use of computer resources is also displayed in the log screen unless you previously specified the NOSTIMER option.
5. **VSE:** If you store your file of display manager commands in an ICCF library, you must assign it the fileref ICCF.
6. **AOS/VS, PRIMOS, and VMS:** The display manager editor breaks information into 80-column lengths; a command that exceeds 80 columns is divided and may not be properly executed.

   CMS, OS, VM/PC, and VSE: Any characters typed beyond the 80th column are truncated.
7. **CMS, OS, VM/PC, and VSE:** You can specify only PROGRAM, LOG, and OUTPUT to indicate an active screen.
8. **AOS/VS, PRIMOS, and VMS:** SASEXEC.SAS can be the actual file name and file type. As long as you have access to the directory in which SASEXEC.SAS is stored, you do not have to assign it a fileref. **Do not** give another file the name SASEXEC.
9. **CMS, OS, VM/PC, and VSE:** 80 columns is the maximum line length in the program editor.

   AOS/VS, PRIMOS, and VMS: 256 columns is the maximum line length in the program editor.
10. **AOS/VS, PRIMOS, and VMS:** The length of the data line on the log screen is sensitive to the value of the LINESIZE= option in the OPTIONS statement. The maximum line length on the log screen is 256 columns.

    CMS, OS, VM/PC, and VSE: The length of the data line on the log screen is sensitive to the value of the TLS= option in the OPTIONS statement. The maximum line length on the log screen is 132 columns.
11. **AOS/VS, PRIMOS, and VMS:** The maximum line length on the output screen is 256 columns.

    CMS, OS, VM/PC, and VSE: The maximum line length on the output screen is 132 columns.

Chapter 15

SAS® Informats and Formats

INTRODUCTION

In the SAS System there are several ways to give directions for reading and writing data values.

A set of directions for reading a value is an **informat**. For example, the BZ. informat directs SAS to read blanks in a numeric field as zeros, instead of ignoring the trailing blanks. SAS provides informats to use to read numeric and character data; date, time, and datetime values; and column-binary data.

A set of directions for writing or printing a value is a **format**. For example, the WORDS. format tells SAS to write a numeric value's equivalent in words. SAS

provides many formats for use with numeric and character data and with date, time, and datetime values. In addition, you can define your own formats with the FORMAT procedure.

Table 15.1 illustrates the techniques available for using SAS informats and formats in particular situations.

Table 15.1 Techniques for Using SAS Informats and Formats

	Techniques	Associate Informats/ Formats with Variables Using...	Use Informat/ Format Directly in...
Reading Values	SAS information (including date, time, datetime, and column-binary informats)	INFORMAT statement	INPUT statement INPUT function
Writing Values	SAS formats (including date, time, and datetime formats) Formats you define with PROC FORMAT	FORMAT statement	PUT statement PUT function

SAS informats and formats have the form

informatw.d
formatw.d

where *informat* or *format* is the name of the informat or format, *w* is a width value (the number of columns in the input or output field), and *d* is an optional decimal scaling factor. If you omit the *w* and *d* values from the informat or format, SAS uses a default *w* value. However, you **must** use a period after the informat or format name.

USING SAS INFORMATS

The simplest way to associate a variable with an informat is to follow the variable with the informat in an INPUT statement. For example, the following INPUT statement:

```
INPUT @15 STYLE 3. @21 PRICE 5.2;
```

uses the *w.* and *w.d* informats, respectively.

You can also associate an informat with a variable in an INFORMAT statement like this one:

```
DATA B;
   INFORMAT BIRTHDAT INTERVW DATE.;
   INPUT @63 BIRTHDAT INTERVW;
```

SAS uses the INFORMAT statement to determine the variable's type (for example, numeric, character, or date) but uses it to determine the length only when it is a character variable. See the chapter "Statements Used in the DATA Step" for more information on the INFORMAT statement.

Note: if you use an informat that is not compatible with the variable's value (for example, alphabetic characters in a numeric field), SAS prints an error mes-

sage on the SAS log describing the problem and sets the variable's value to missing. See **Illegal Characters in Input Data** in the "Missing Values," chapter for more information.

Table 15.2 SAS Numeric and Character Informats

SAS Numeric Informats				
Informat	Description	Width Range	Decimal Range	Default Width
w.	standard numeric	1-32		
w.d			0-31	
BZw.d	blanks are zeros	1-32	0-31	1
COMMAw.d	commas in numbers	1-32	0-31	1
Ew.d	scientific notation	7-32	0-31	12
HEXw.	numeric hexadecimal	1-16		8
IBw.d	integer binary	1-8	0-10	4
PDw.d	packed decimal	1-16	0-10	1
PIBw.d	positive integer binary	1-8	0-10	1
PKw.d	unsigned packed decimal	1-16	0-10	1
RBw.d	real binary (floating point)	2-8	0-10	4
ZDw.d	zoned decimal AOS/VS, PRIMOS, VMS:	1-16	0-10	1
	CMS, OS, VSE:	1-32	0-10	1
ZDBw.d	zoned decimal with blanks legal	1-32	0-10	1
SAS Character Informats				
$w.	standard character	1-200		1 or length of variable
$CHARw.	characters with blanks	1-200		1 or length of variable
$CHARZBw.	characters with binary zeros as blanks	1-200		1 or length of variable
$HEXw.	character hexadecimal	1-200		2
$PHEXw.	packed hexadecimal as character data	1-100		2
$VARYINGw.	varying-length values	1-200		8 or length of variable

Additional informats are described in **Using SAS Date, Time, and Datetime Informats and Formats** later in this chapter.

Informat Descriptions: Numeric

w.d informat: standard numeric data
range: 1-32

The standard SAS numeric informat *w.d* can read standard numeric data values (one digit per byte), where

w is a number giving the length, in columns, of the field containing the value.

d is an optional number giving the number of digits to the right of the decimal point in the value.

The *w.d* informat can read numeric values located anywhere in the field; blanks can precede or follow the value. A minus sign precedes negative values, without a blank between the sign and the value. Values read with the *w.d* informat can include decimal points. Values in scientific E-notation can be read with either the *w.d* informat or the *Ew.d* informat, described below.

Include a *d* value in the *w.d* informat when you want the SAS System to insert decimal points in numeric input values entered without decimal points. When an informat includes a *d* value and the data value does not include a decimal point, the value is divided by 10^{d}. (However, a decimal point already in the data value remains in its original position.)

Note that with the *w.d* informat, trailing blanks are not the same as trailing zeros. (If you want trailing blanks to be read as zeros, use the BZ. informat, below.) In addition, the $w. informat converts a period (.) to a blank, because it interprets the period as a missing value.

Example 1 The following DATA step shows several representations of the number 23, all of which can be read with the numeric informat 6.:

```
DATA A;
   INPUT @1 X 6.;
   CARDS;
23         left-aligned
    23     right-aligned
  23       in the middle
  23.0     with decimal point
 2.3E1     in scientific notation
  -23      negative value
```

Example 2 The data lines below show four representations of the number 23, each of which can be read with the informat 6.2:

```
DATA A;
   INPUT @1 X 6.2;
   CARDS;
  2300    right-aligned
2300      left-aligned
 -2300    negative value
   23.    explicit decimal point
```

Using column input is equivalent to using a pointer control and a *w.d* informat. For example, the statement

```
INPUT X 1-6 .2;
```

is equivalent to

```
INPUT @1 X 6.2;
```

Do not use the BZ. informat in an INFORMAT statement within a DATA step if the INPUT statement uses list input because SAS interprets blanks between values in the data line as delimiters rather than zeros. For example, do not use statements like the following:

```
INFORMAT x BZ5.;
INPUT X Y Z;
```

BZw.d informat: blanks are zeros
range: 1-32
default: 1

Sometimes data are entered without punching embedded and trailing zeros. For example, say that the value 340 is entered in columns 2-4; however, instead of zero in column 4, there is a blank. When FORTRAN reads this value, it translates the blank to zero; however, the *w.* and *w.d* SAS informats read the value as 34.

To read the value as 340, use the BZ. (**B**lanks are **Z**eros) informat. The BZ. informat is identical to the standard *w.d* informat except that it treats blanks other than leading blanks as zeros.

Example 1 X's value is 340 but is entered as 34 followed by a blank.

EBCDIC hex: F3 F4 40
ASCII hex: 33 34 20
input data: 34
informat: BZ3.

```
INPUT @10 X BZ3.;
```

Example 2 Y's value is −200 but is entered as '−2 '.

EBCDIC hex: 60 F2 40 40
ASCII hex: 2D 32 20 20
input data: -2
informat: BZ4.

```
INPUT @ 5 Y BZ4.;
```

COMMA.w.d informat: embedded characters

The COMMA informat removes embedded commas, blanks, dollar signs, percent signs, and parentheses from input data. Parentheses at the beginning of a field are converted to minus signs.

Ew.d informat: scientific notation
range: 1-32
default: 1

You can read values of numeric variables in scientific notation using the *Ew.* informat. However, the *E.* informat is normally not needed since the standard *w.d* numeric informat can read numbers in scientific notation.

Example X's value is 1.257E3, stored in columns 10-16 of a data line.

informat: E7.

```
INPUT @10 X E7.;
```

HEXw. informat: numeric hexadecimal
range: 1-16
default: 8

The HEXw. informat converts a hexadecimal value to fixed point binary. Each two columns of hex digits produce one byte of corresponding fixed point binary.

When HEX16. is specified, the value is read as the floating-point representation of the number, rather than the fixed-point representation.

Example Columns 21-25 contain the value 1003A, which is the hexadecimal, fixed-point representation of the decimal number 65594.

EBCDIC hex: F1 F0 F0 F3 C1
ASCII hex: 31 30 30 33 41
informat: HEX5.

```
INPUT @21 X HEX5.;
```

IBw.d informat: integer binary
range: 1-8
default: 4

The IB informat reads fixed-point binary values. For integer binary data, the high-order bit is the value's sign: 0 for positive values, 1 for negative. Negative values are represented in two's complement notation. If the informat includes a *d* value, the number is divided by 10^d. It is usually impossible to key in binary data directly from a terminal, though many programs write data in binary. The notation for integer binary in several programming languages is as follows:

SAS	IB2.	IB4.
PL/I	FIXED BIN(15)	FIXED BIN(31)
FORTRAN	INTEGER*2	INTEGER*4
COBOL	COMP PIC 9(4)	COMP PIC 9(8)
IBM assembler	H	F

Example 1 The value 128 is stored as a 4-byte integer binary number in columns 20-23 of a data line.

hex: 00 00 00 80
informat: IB4.

```
INPUT @20 X IB4.;
```

Example 2 The value −255 is stored in columns 20-23 of the data line:

hex: FF FF FF 01
informat: IB4.

```
INPUT @10 X IB4.;
```

PDw.d informat: packed decimal
range: 1-16
default: 1

Each byte contains two digits in packed decimal data. The value's sign is in the last half of the last byte: a C or F if the value is positive, a D if it is negative. Although it is usually impossible to key in packed decimal data directly from a terminal, many programs write data in packed decimal. The notation for packed decimal in several programming languages is as follows:

SAS	PD4.
PL/I	FIXED DEC(7,0)
COBOL	COMP-3 PIC S9(7)
IBM assembler	PL4

Example 1 The value 128 is stored in packed decimal form in columns 4-7 of a data line.

hex: 00 00 12 8C
informat: PD4.

```
INPUT @4 X PD4.;
```

Example 2 You have a packed decimal date value from which you want to create a SAS date variable:

```
DATA NEW;
   INPUT MNTH PD4.;
   DATE=INPUT(PUT(MNTH,6.),MMDDYY6.);
```

The PUT function converts the packed decimal value to standard numeric—the way it looks when printed. Then the INPUT function uses the MMDDYY informat to read that value as a SAS date value.

PIBw.d informat: positive integer binary
range: 1-8
default: 1

Positive integer binary values are the same as integer binary (see the IB. informat, above), except that all values are treated as positive. Thus, the high-order bit is part of the value rather than the value's sign.

If the informat includes a *d* value, the data value is divided by 10^{d}.

If you are planning to test the bits of a byte, read the value with the PIB1. informat. Another way is to read it with the $1. informat.

When you want the decimal equivalent of one of the 256 EBCDIC or 128 ASCII characters, read the character with the PIB. informat. For example, the hex code for the letter A is C1 in EBCDIC with a decimal equivalent of 193, and 40 in ASCII with a decimal equivalent of 65. The statements

```
DATA X;
   INPUT CHAR PIB1.;
   PUT CHAR;
   CARDS;
A
;
```

produce 193 in EBCDIC and 65 in ASCII.

Example The value 12 is stored as a 1-byte positive integer binary value in column 43 of a data line.

hex: 0C
informat: PIB1.

```
INPUT @43 X PIB1.;
```

PKw.d informat: unsigned packed decimal
range: 1-16
default: 1

The PK. informat is like the PD. informat described above, except that all values are positive, and the last half of the last byte contains a digit rather than the value's sign. W represents the number of bytes, not the number of digits.

Example The value 128 is stored in packed decimal form in columns 5-7 of a data line.

hex: 00 01 28
informat: PK3.

```
INPUT @5 X PK3.;
```

RBw.d informat: real binary (floating point)
range: 2-8
default: 4

Numeric data for scientific calculations are often stored in floating-point representation. (The SAS System stores all numeric values in floating-point.) A floating-point value consists of two parts: a mantissa giving the value and an exponent giving the value's magnitude. It is usually not possible to key in floating-point binary data directly from a terminal, though many programs write data in floating point binary. This table compares the names of floating point notation in several languages:

	4 bytes	8 bytes
SAS	RB4.	RB8.
PL/I	FLOAT BIN(21)	FLOAT BIN(53)
FORTRAN	REAL*4	REAL*8
COBOL	COMP-1	COMP-2
IBM assembler	E	D

Example The value 128 as an 8-byte floating point value is stored in columns 10-17 of a data line.

hex: 41 80 00 00 00 00 00 00
informat: RB8.

```
INPUT @10 X RB8.;
```

ZDw.d informat: zoned decimal
range: AOS/VS, PRIMOS, VMS: 1-16
CMS, OS, VSE: 1-32
default: 1

Zoned decimal is similar to standard numeric informat in that every digit requires one byte. However, the value's sign is in the last byte along with the last digit. If the value is positive, the next-to-last hex digit is A, C, E, or F; if the value is negative, the next-to-last hex digit is B or D. Positive values can be entered in zoned decimal from a terminal. Some keying devices allow negative values to be entered by overstriking the last digit with a minus sign. The following table compares the zoned decimal informat with notation in several languages:

SAS	ZD3.
PL/I	PICTURE'99T'
COBOL	DISPLAY PIC S 999
IBM assembler	ZL3

Example The value 128 is stored in zoned decimal form in columns 4-7 of a data line.

hex: F0 F1 F2 C8
informat: ZD4.

```
INPUT @4 X ZD4.;
```

ZDBw.d informat: zoned decimal with blanks
range: 1-32
default: 1

The ZDB. informat reads zoned decimal data produced in IBM 1410, 1401, and 1620 form, in which zeros are left blank rather than being punched. The *w* value is the number of columns of the field containing a value, and the optional *d* value gives the number of decimal places for the value. The *d* value may be larger than the *w* value.

Example The zoned decimal value 102 is stored in columns 10-12 of a data line, with blanks representing zeros.

EBCDIC hex: F1 40 C2
informat: ZDB3.

```
INPUT @10 X ZDB3.;
```

Informat Descriptions: Character

$w. informat: standard character data
range: 1-200
default: 1 if the length of the variable is not yet defined;
otherwise, the length of the variable.

Use the SAS informat *$w.* to read character data. The *w* value gives the number of columns in the field containing the character value. The *$w.* informat trims leading blanks before storing values; that is, it automatically left aligns when reading values.

Example Columns 10-12 of the input data line contain the value 'ABC'.

EBCDIC hex: C1 C2 C3
ASCII hex: 41 42 43
informat: $3.

The statement

```
INPUT @10 NAME $3.;
```

reads the value as

'ABC'

This statement is equivalent to

```
INPUT NAME $ 10-12;
```

Example 2 Columns 21-25 of an input data line contain the value ' XYZ'.

EBCDIC hex: 40 40 E7 E8 E9
ASCII hex: 20 20 58 59 5A
informat: $5.

```
INPUT @21 NAME $5.;
```

reads the value as

'XYZ '

$CHARw. informat: leading and trailing blanks
range: 1-200
default: 1 if the length of the variable is not yet defined;
otherwise, the length of the variable

The $CHARw. informat is identical to the *$w.* informat above, except that $CHAR. does not trim leading blanks. This table compares the SAS informat $CHAR8. with notation in other languages:

SAS	$CHAR8.
PL/I	CHAR(8)
FORTRAN	A8
COBOL	PIC X(8)
assembler	CL8

Example The value ' ABC' with one leading blank is in columns 7-10 of the data line.

EBCDIC hex: 40 C1 C2 C3
ASCII hex: 20 41 42 43
informat: $CHAR4.

The statement

```
INPUT @7 NAME $CHAR4.;
```

reads this value as

' ABC'

Do not use the $CHAR. informat in an INFORMAT statement within a DATA step if the INPUT statement uses list input because SAS interprets blanks between values in the data line as delimiters. For example, do not use statements like the following:

```
INFORMAT X $CHAR12.;
INPUT X Y Z;
```

$CHARZBw. informat: binary zeros as blanks
range: 1-200
default: 1 if the length of the variable is not yet defined;
otherwise, the length of the variable

The $CHARZB. informat is identical to the $CHARw. informat, except that $CHARZB. reads the specified input area and changes any byte of binary zero (X'00') to a blank character (X'40'). Binary zeros instead of blanks are occasionally found in various account code and programmer name fields of OS/360 and OS/VS SMF records and accounting data.

Example The value 'SMITH, JOHN ' is in a data record at offset +42 with trailing binary zeros instead of blanks.

EBCDIC hex: E2 D4 C9 E3 C8 6B 40 D1 D6 C8 D5 00 00 00 00 00
00 00 00 00
ASCII hex: 53 4D 49 54 48 2C 20 4A 4F 48 4E 00 00 00 00 00
00 00 00 00
informat: $CHARZB20.

```
INPUT @43 NAME $CHARZB20.;
```

$HEXw. informat: character hexadecimal
range: 1-200
default: 2

The $HEX. informat is like the HEX. informat in that it reads values in which each hex digit occupies one byte. Use the $HEX. informat to encode binary information into a character variable when your input data are limited to printable characters.

Example Columns 21-24 contain the value C1C2.

EBCDIC hex: C3 F1 C3 F2
ASCII hex: 43 31 43 32
informat: $HEX4.

```
INPUT @21 NAME $HEX4.;
```

In the SAS data set, the character variable NAME has a length of 2, even though the *w.* value in the informat is 4, since SAS stores the 4 hex digits C1C2 in 2 bytes.

$PHEXw. informat: packed hexadecimal as character data
range: 1-100
default: 2

The $PHEX. informat converts packed hexadecimal data to character data. Packed hexadecimal data are like packed decimal data, except that all hex digits are valid and the value of the low-order nibble (which would indicate the sign in the case of packed decimal data) is ignored. Unlike the packed decimal informat, however, the $PHEX. informat returns a character value, and the value of the sign nibble is treated as if it were X'F', regardless of its actual value. Packed hexadecimal can be found in SMF record type 74 (RMF Device Activity) field SMF74ADD.

Example The two bytes at offset +4 in a record contain the hexadecimal value '1E0F'.

hex: 1E 0F
informat: $PHEX2.

```
INPUT @5 DEVADDR $PHEX2.;
```

In the SAS data set, the character variable DEVADDR has a length of 3, even though the *w.* value in the informat is 2, since SAS requires 3 bytes to store the character representation (X'F1C5F0') of the 3 hexadecimal digits 1E0. If the value is printed with the standard SAS character format, it prints as '1E0'. In general, a character variable whose length is implicitly defined with the $PHEXw. informat has a length of 2w−1.

$VARYINGw. informat: varying-length values
range: 1-200
default: 8 or length of variable

Use the $VARYINGw. informat when the length of a character value differs from record to record. Typically, the character value's length in the current record is given in a numeric field in the record (or implicitly by the fact that the character variable occupies the entire varying portion of a variable-length record). A numeric *w* value specifying the character variable's maximum length usually follows the $VARYING. informat.

With the $VARYINGw. informat, SAS first determines a character value's length in the current observation from a numeric length variable. It can obtain the value in several ways, such as by reading a field described in the same INPUT statement as the character variable, by reading the length field in another INPUT statement, or by calculating the value. SAS then reads the character value for that observation from the number of columns specifed. The variable's value is padded with blanks if necessary to equal the maximum length specified by *w* before being stored.

The $VARYINGw. informat **must** include a length specification.[1]

If the length variable's value is 0 or negative (including missing values) for a given observation, SAS assigns it a length of 1. If the length variable's value is greater than 0 but less than the *w* value, SAS reads the number of columns specified by the length variable, and pads the value with trailing blanks to the maximum length. If the value of the length variable is greater than the *w* value, SAS uses the *w* value for the character variable's length. The length variable cannot be an array name.

The SAS System always associates a *w* value with the $VARYING. informat. If you do not specify a value for *w*, SAS assigns a length to the character variable and uses that length as the value of *w*. SAS assigns the character variable's length with the same rules it uses when assigning lengths to character variables in other situations: if you have given the variable a length earlier (for example, with a length statement) it uses that length; if you have not given the variable a specific length, it uses a length of 8.

The pointer's position after reading a data value with the $VARYINGw. informat is the first column after the value.

range 1-200
default 8 or length of variable

Example DSNAME is a character variable whose length can vary from one to forty-four characters.

In the input lines, the LENVAR variable contains DSNAME's length, in bytes, for the current line. The INPUT statement first reads the LENVAR variable and then uses it after the $VARYING44. informat to give the current length of the DSNAME value, up to a maximum of forty-four characters:

```
DATA TEST;
   INPUT LENVAR PIB1. DSNAME $VARYING44. LENVAR;
```

USING SAS FORMATS

The simplest way to associate a format with a variable is to follow the variable with a format in the PUT statement. The following example uses the DOLLAR. format to write numeric values as dollar amounts:

```
DATA MONEY;
```

```
AMOUNT=1145.32;
PUT AMOUNT DOLLAR10.2;
```

The statements above produce the value

```
$1,145.32
```

In DOLLAR10.2, the *w* value of 10 and the *d* value of 2 indicate a maximum of 10 columns for the value, with two of these columns for the decimal part of the value, one for the decimal point, and seven reserved for the minus sign (if the value is negative), dollar sign, comma, and dollar part of the value.

A FORMAT statement also associates a format with a variable. However, there is a difference between using a format in the PUT statement as opposed to specifying the format in a FORMAT statement. A format in the PUT statement preserves leading blanks, whereas a FORMAT statement trims leading blanks.

If you want to associate a format with a variable permanently so that later PROC and DATA steps use the format, you **must** specify the variable and format in a FORMAT statement in a DATA step. For example,

```
DATA WHOLEYR;
   INPUT SALES1-SALES12;
   SALESYR=SUM(OF SALES1-SALES12);
   FORMAT SALES1 SALES7 SALESYR DOLLAR10.;
PROC PRINT;
   VAR SALES1 SALES7 SALESYR;
```

The PRINT procedure prints the variables SALES1, SALES7, and SALESYR preceded by dollar signs and with commas separating each three columns of figures.

If you want to associate a format with a variable only for the duration of a procedure, use a FORMAT statement in the PROC step. For example, the statements

```
PROC PRINT DATA=PRODUCTS;
   FORMAT PRICE DOLLAR8.2;
   VAR STYLE COLOR PRICE;
```

print the values of PRICE with dollar signs, commas, and two decimal places.

If a variable is specified in more than one FORMAT statement, the format given in the last FORMAT statement is used. To delete a variable's format, use the variable name in a FORMAT statement without a format. You can also change or delete formats with PROC DATASETS. See "The DATASETS Procedure" for more information.

If the value of a variable does not fit into the width of the format you are using, SAS tries to squeeze the value into the space available. Character formats truncate values on the right. Numeric formats sometimes revert to the BEST. format. The SAS System prints asterisks if it is not possible to represent the value.

The preceding information on SAS formats applies both to formats used in the DATA step and those created with PROC FORMAT.

Formats created with PROC FORMAT can be stored for use in later jobs.[2]

Note: storing formats is an important consideration when you are associating formats with variables for permanent SAS data sets, especially when the data sets are shared with other users. When the FMTERR option is in effect, SAS produces an error message if it cannot find the format. When the system option NOFMTERR is in effect, a variable associated with a format SAS cannot find is processed with a default format (usually w. or $w.). Although the NOFMTERR option allows SAS to work with the variable, you lose the information supplied by the format. To avoid losing information in formats you have created, always give directions for accessing the stored formats with directions for accessing the data set. Or, if you ship a tape containing a SAS data set with user-created formats

to a SAS user at another installation, be sure that one file on the tape contains the formats.[3]

Table 15.3 SAS Numeric and Character Formats

SAS Numeric Formats					
Format	Description	Width Range	Decimal Range	Default Width	Alignment
w.	standard numeric	1-32			right
w.d			d<w		
BESTw.	SAS chooses best notation	1-32		12	right
COMMAw.d	commas in numbers	2-32	0 or 2	6	right
DOLLARw.d	dollar sign, commas	2-32	0 or 2	6	right
Ew.	scientific notation	7-32		12	right
FRACTw.	fractions	4-32		10	right
HEXw.	numeric hexadecimal	1-16		8	left
IBw.d	integer binary	1-8	0-10	4	left
PDw.d	packed decimal	1-16	0-10	1	left
PIBw.d	positive integer binary	1-8	0-10	1	left
RBw.d	real binary (floating point)	2-8	0-10	4	left
ROMANw.	Roman numerals	2-32		6	left
SSNw.	social security numbers	11		11	
WORDFw.	numbers as words with fractions as numbers	5-200		10	left
WORDSw.	numbers as words with fractions as words	5-200		10	left
Zw.d	print leading zeros	1-32		1	right
ZDw.d	zoned decimal	1-16	0-10	1	left

Continued on next page

Table 15.3 *Continued*

SAS Character Formats				
Format	Description	Width Range	Default Width	Alignment
$w.	standard character	1-200	1 or length of variable	left
$CHARw.	characters with blanks	1-200	1 or length of variable	left
$HEXw.	character hexadecimal	1-200	2	left
$VARYINGw.	varying-length character values	1-200	8 or length of variable	left

Additional formats are described in **Using SAS Date, Time, and Datetime Informats and Formats** later in this chapter.

Format Descriptions: Numeric

W.d format: standard numeric data
range: 1-32
default: 1

You can use the *w.d* format to write values in a field *w* positions wide, with *d* positions to the right of the decimal point. If *d* is 0 or if it is omitted, the value is written without a decimal point.

Numbers written with the *w.d* format are rounded to the nearest number that can be represented in the output field. If the number is too large to fit, the BEST. format, described below, is used instead. Negative numbers are printed with a leading minus sign.

In choosing a *w* value, allow enough space to write the value, the decimal point, and a minus sign if necessary.

Example The statements

```
DATA;
   X=23.45;
   PUT X 6.3;
```

produce the line

```
 23.450
```

Using column output is equivalent to using a pointer control and a *w.d* format. For example, the statement

```
PUT X 1-8 .2;
```

is equivalent to

```
PUT @1 X 8.2;
```

BESTw. format: SAS chooses best notation
range: 1-32
default: 12
alignment: right

The BEST. format is the default when a format is not specified. SAS chooses the notation that gives the most information about the value that will fit into the number of columns available.

Example 1 X's value is 1257000, and you want to write it in the six columns 10-15. Since seven columns are needed to represent the value exactly, SAS squeezes the value into E-notation to get it into six columns: the statement

```
PUT @10 X BEST6.;
```

prints the value

```
1.26E6
```

Example 2 X's value is 1257000, and you want to write it in columns 10-12. The SAS System puts the most information possible into three columns: the statement

```
PUT @10 X BEST3.;
```

prints the value

```
1E6
```

Although part of the value is lost, the value still prints. If you give only two columns to print this value, SAS prints asterisks instead of trying to represent the value.

COMMAw.d format: commas
range: 2-32
default: 6
alignment: right

The COMMA. format is like the DOLLAR. format, below, except that it does not print a dollar sign in front of the value. You might use DOLLAR. to print a total, and use COMMA. to print the detail lines. The *d* value, if specified, must be either 0 or 2.

Example SALES' value is 23451.23, and you want to print it in columns 24-33 of the output lines.

```
PUT @24 SALES COMMA10.2;
```

prints the value

```
23,451.23
```

DOLLARw.d format: dollar sign, commas, and decimal point
range: 2-32
default: 6
alignment: right

You can print numeric values as dollar amounts with the DOLLAR. format. A dollar sign precedes the value, commas separate every three digits, and if the format includes a decimal value, two decimal digits representing cents are printed following a decimal point.

The *d* value, if specified, must be either 0 or 2. If the value is too large for the field, the BEST. format is used instead.

Example NETPAY's value is 1254.71, and you want to print it in columns 53-62 of the output line.

```
PUT @53 NETPAY DOLLAR 10.2;
```

prints the value

```
$1,254.71
```

Ew. format: scientific notation
range: 7-32
default: 12
alignment: right

You can write numeric variable values in scientific notation using the E. format.

Example X's value is 1257, and you want to write it in columns 10-19 of the output line in scientific notation.
format: E10.

```
PUT @10 X E10.;
```

prints the value

```
1.257E+03
```

Column 10 will be blank because it is reserved for a minus sign.

FRACTw. format: fractions
range: 4-32
default: 10
alignment: right

The FRACT. format converts values to fractions. For example, it is common in matrix operations to divide the number 1 by 3 to produce the value .33333333. If you prefer to print this value as 1/3, use the FRACT. format. FRACT. prints fractions in reduced form, that is, 1/2 instead of 50/100.

Example 1 X's value is .6666666667, and you want to print it in columns 13-15 of the output lines.

```
PUT @13 X FRACT3.;
```

prints the value

```
2/3
```

Example 2 Y's value is .2784.

```
PUT Y FRACT.;
```

The PUT statement prints the value

```
174/625
```

HEXw. format: numeric hexadecimal
range: 1-16
default: 8
alignment: left

The HEXw. format converts a number's fixed-point binary representation to hexadecimal. Each byte requires two columns to represent the corresponding hex digits.

When HEX16. is specified, the number is represented in floating-point rather than fixed-point.

Example X's value is 100, and you want to print the hex equivalent in columns 2 and 3 of the data line:

```
PUT @2 X HEX2.;
```

prints the value 64.

IBw.d format: integer binary
range: 1-8
default: 4 alignment: left

Integers are stored in integer binary, or fixed-point, form. For example, the number 2 is stored as 00000002. If the format includes a *d* value, the number is divided by 10^d. The following table compares integer binary notation in several languages:

SAS	IB2.	IB4.
PL/I	FIXED BIN(15)	FIXED BIN(31)
FORTRAN	INTEGER*2	INTEGER*4
COBOL	COMP PIC 9(4)	COMP PIC 9(8)
assembler	H	F

Example You want to write the numeric variable X's value of 128 in columns 20-23 of the output line:

```
PUT @20 X IB4.;
```

PDw.d format: packed decimal
range: 1-16
default: 1
alignment: left

In packed decimal data, each byte except the last contains two digits. The *w* value represents the number of bytes, not the number of digits. The value's sign is in the last half of the last byte: a C or F if the value is positive, a D if it is negative. This table compares the SAS format PD4. with notation in some other languages:

SAS	PD4.
PL/I	FIXED DEC(7,0)
COBOL	COMP-3 PIC S9(7)
assembler	PL4

Example X's value is 128, and you want to store it in packed decimal form in columns 4-7 of an output line.

```
PUT @4 X PD4.;
```

PIBw.d format: positive integer binary
range: 1-8
default: 1
alignment: left

Positive integer binary values are the same as integer binary (see the IB. format, above), except that all values are treated as positive. Thus, the high-order bit is part of the value rather than the value's sign.

If a *d* value is specified in the PIB. format, the data value is divided by 10^d.

Example X's value is 12, and you want to write it as a 1-byte positive integer binary value in column 43 of an output line:

```
PUT @43 X PIB1.;
```

RBw.d format: real binary (floating point)
range: 2-8
default: 6
alignment: left

Numeric data for scientific calculations are commonly represented in floating point. (SAS stores all numeric values in floating point.) A floating point value consists of two parts: a mantissa giving the value, and an exponent giving the value's magnitude.

Real binary is the most efficient format for representing numeric values, since SAS already represents numbers this way and no conversion is needed. This table compares the names of floating point notation in several languages:

	4 bytes	8 bytes
SAS	RB4.	RB8.
PL/I	FLOAT BIN(21)	FLOAT BIN(53)
FORTRAN	REAL*4	REAL*8
COBOL	COMP-1	COMP-2
assembler	E	D

Example The numeric variable X's value is 128, and you want to write it in columns 10-17 of the output line:

```
PUT @10 X RB8.;
```

ROMANw. format: Roman numerals
range: 2-32
default: 6
alignment: left

The ROMAN. format prints numeric values as Roman numerals. Noninteger values are truncated to integers before printing.

Example X's value is 1982, and you want to print it in columns 25-35 of the output line.

```
PUT @25 X ROMAN10.;
```

prints the value

```
MCMLXXXII
```

SSNw. format: social security numbers
range: 11
default: 11

The SSN. format prints 9-digit numeric values as U.S. social security numbers, with dashes between the third and fourth digits and between the fifth and sixth digits. If the value is missing, SAS prints nine single periods with dashes between the third and fourth period and between the fifth and sixth periods. If the value contains fewer than nine digits, SAS aligns the value on the right and pads the value with zeros on the left. If the value has more than nine digits, it is set to missing.

Example: ID's value is 263878439, and you want to print it in columns 21-31 of the output line:

```
PUT @21 ID SSN11.;
```

prints the value

```
263-87-8439
```

WORDFw. format
range: 5-200
default: 10
alignment: left

The WORDF. format converts numeric values to their equivalent in words. It is identical to the WORDSw. format except that fractions print as numbers instead of words. Fractions are represented as hundredths. For example, 8.2 prints as EIGHT AND 20/100. Negative numbers are preceded by the word MINUS. When the value's equivalent in words does not fit into the specified field, it is truncated on the right and the last character prints as an asterisk.[4]

Example PRICE's value is 29.95, and you want to print it in columns 40-69 of the output line.

```
PUT @40 PRICE WORDF30.;
```

prints the value

```
TWENTY-NINE AND 95/100
```

WORDSw. format
range: 5-200
default: 10
alignment: left

The WORDS. format converts numeric values to their equivalent in words. For example, you might want to print checks with the amount written out below the payee line.

Negative numbers are preceded by the word MINUS. If the number is not an integer, the fractional portion is represented as hundredths. For example, 5.3 prints as FIVE AND THIRTY HUNDREDTHS. When the value's equivalent in words does not fit into the specified field, it is truncated on the right and the last character prints as an asterisk.[5]

Example NETPAY's value is 354, and you want to print it in columns 20-69 of the output line.

```
PUT @20 NETPAY WORDS50.;
```

prints the value

```
THREE HUNDRED FIFTY-FOUR
```

Zw.d format: print leading zeros
range: 1-32
default: 1
alignment: right

The Z. format fills with zeros rather than blanks to the left of the data value.

Example SEQNUM's value is 1350, and you want to print it in columns 73-80 of the output line, with zeros in the columns before the value.

```
PUT @73 SEQNUM Z8.;
```

prints the value

```
00001350
```

ZDw.d format: zoned decimal
range: 1-32
default: 1
alignment: left

Zoned decimal is similar to standard numeric format in that every digit requires one byte. However, the value's sign is in the last byte along with the last digit. If the value is positive, the next-to- last hex digit is C. If the value is negative, the next-to-last hex digit is D. This table compares the zoned decimal format with notation in several languages:

SAS	ZD3.
PL/I	PICTURE '99T'
COBOL	DISPLAY PIC S 999
assembler	ZL3

Example X's value is 102, and you want to write it in zoned decimal format in columns 4-7 of the output line:

```
PUT @4 X ZD4.;
```

Format Descriptions: Character

range: 1-200
default: 1 if length of variable not yet defined; otherwise,
the length of the variable
alignment: left

To write a character variable, you must use a character format. Character formats begin with a dollar sign ($).

$w. format: standard character data

To write character data, use the SAS format $w. The *w* value gives the number of columns used to write the character value.

Example NAME's value is 'ABC', and you want to write it in columns 10-12 of the output line.

```
PUT @10 NAME $3.;
```

or

```
PUT NAME $ 10-12;
```

or

```
PUT NAME 10-12;
```

You can omit the dollar sign in a PUT statement, since SAS knows that NAME is a character variable:

$CHARw. format: leading blanks
range: 1-200
default: 1 if length of variable not yet defined; otherwise, the length of the variable
alignment: left

The $CHARw. format is identical to the $w. format above, except that $CHAR. does not trim leading blanks. This table compares the $CHARw. format to notation in some other languages:

SAS	$CHAR8.
PL/I	CHAR(8)
FORTRAN	A8
COBOL	PIC X(8)
assembler	CL8

Example NAME's value is ' ABC', with one leading blank, and you want to write it in columns 7-10 of an output line.

```
PUT @7 NAME $CHAR4.;
```

$HEXw. format: character hexadecimal
range: 1-200
default: 2
alignment: left

The $HEX. format is like the HEX. format, above, in that it converts a character value to hexadecimal, with each byte requiring two columns.

Example The character variable NAME's value is 'AB', and you want to print its hex equivalent in columns 21-24. Note that 4 columns are required for the output value.

```
PUT @21 NAME $HEX4.;
```

$VARYINGw. format: varying-length values
range: 1-200
default: 8 or length of variable
alignment: left

Use the $VARYINGw. format when the length of a character value differs from record to record. Typically, the character value's length in the current record is given in a numeric field in the record (or implicitly, by the fact that the character variable occupies the entire varying portion of a variable-length record). A numeric *w* value specifying the variable's maximum length usually follows the $VARYING. informat.

With the $VARYINGw. format, SAS ascertains the character value's length to determine how long the printed value is to be. The length variable is not printed. The $VARYINGw. format **must** include a length specification.[6]

If the length variable's value is 0 or negative (including missing values) for a given observation, SAS assigns it a length of 1. If the length variable's value is greater than 0 but less than the *w* value, SAS prints the length specified by the length variable. If the value of the length variable is greater than the *w* value, SAS uses the *w* value for the character variable's length. The length variable cannot be an array name.

The SAS System always associates a *w* value with the $VARYING. format. If you do not specify a value for *w*, SAS assigns a length to the character variable and uses that length as the value of *w*. SAS assigns the character variable's length with the same rules that it uses when assigning lengths to character variables in other situations: if you have given the variable a length earlier (for example, with a LENGTH statement) it uses that length; if you have not given the variable a specific length, it uses a length of 8. Note: do not use the $VARYINGw. format in a FORMAT statement within a PROC step because you may get unexpected results.

The pointer's position after printing a value with the $VARYINGw. format is the first column after the value.

Example CITY is a variable in your data set whose length ranges from 1 to 30 characters, although it is stored with a length of 30. LEN is another variable in the data set giving the actual length of CITY for the current observation. You want to write the CITY value beginning in column 10; the pointer's position after writing the value should be the first column after the actual CITY value.

```
PUT @10 CITY $VARYING30. LEN;
```

Note that the LEN variable in this statement is part of the $VARYING30. format.

USING SAS DATE, TIME, AND DATETIME INFORMATS AND FORMATS

You can read, work with, and write values that represent dates and times in SAS. SAS represents each of these values as a number associated with an implicit time unit:

- a date in SAS is represented by the number of days between January 1, 1960 and that date
- a time in SAS is represented in seconds
- a date and time is represented by the number of seconds between midnight, January 1, 1960, and the date and time.

For example, consider the date July 4, 1982. Suppose that this date is written in an input data line as 7-4-82. To read the value as a SAS date value, use the date informat MMDDYY8., which reads 7-4-82 and converts it to the number of days between January 1, 1960 and July 4, 1982: 8220 days.

```
DATA DAYS;
   INPUT BIRTHDAY MMDDYY8.;
   CARDS;
7-4-82
```

After the INPUT statement executes, BIRTHDAY's value is 8220, and if you print it with the statement

```
PUT BIRTHDAY;
```

the value 8220 prints.

In order to print 8220 as a date, you must assign a format to the variable BIRTHDAY. There are several formats available that print the date. One is DATEw., which prints the date as the day of the month, followed by the first three letters of the month, and then the year. So the statement

```
PUT BIRTHDAY DATE7.;
```

prints the line

```
04JUL82
```

You can associate a format with the value for the duration of the job with a FORMAT statement in the DATA step. For example, the statement

```
FORMAT BIRTHDAY DATE7.;
```

prints the BIRTHDAY value as 04JUL82 for the remainder of the job.

This representation of dates and times has several advantages: date, time, and datetime values sort correctly; finding intervals between two dates or two times involves only a simple subtraction; and the standards are internationally recognizable.

SAS dates are valid back to A.D. 1582 and ahead to A.D. 20,000, and leap year, century, and fourth-century adjustments are handled properly. However, leap seconds are ignored, and the SAS System does not adjust for daylight saving time.

If the width of your informat is not sufficient to read all the columns containing the date, time, or datetime value in your data lines, you will get unexpected results. (Remember that blanks or special characters between the day, month, year, or time add to the length of the value.) For example, if you use the informat DATE8. to read these values:

```
01/JAN/82
3/MAR/1955
```

SAS assigns date values corresponding to 01JAN08 and 03MAR19, respectively. If the width is such that the last column SAS reads is a special character, SAS produces an invalid data message.

If the width of a date, time, or datetime format is not sufficient to write all the information about the value, SAS truncates the value on the right. In some formats SAS abbreviates the name of the month, the name of the weekday, or both, in order to print as much information as possible. If the format specifies more columns than necessary, the value is right aligned.

If you give a two-digit year value, SAS assumes that the year is in the 1900's.

Duration vs. Date

When you are working with date and datetime values, it is important to remember the associated units when manipulating the values.

For example, suppose you want to calculate the ages of employees from their birth dates. You read the birth dates with a date informat (say DATE7.), and use the function TODAY to assign today's date to a variable. To find the employees' ages, subtract BIRTHDAY from TODAY:

```
DATA EMPLAGE;
   INPUT ID BIRTHDAY DATE7.;
   TODAY=TODAY();
   AGE=TODAY-BIRTHDAY;
```

The value of AGE is the number of days between the employees' birthdays and today, representing a duration in days rather than in years.

As an example, suppose an employee's birthday is December 1, 1959. The internal representation of that date is −31. Today's date is January 15, 1983; the internal representation of that date is 8415. Subtracting −31 from 8415 gives 8446, which is the number of days between today and December 1, 1959. This number is **not** a SAS date value since it represents a duration that is not based on January 1, 1960. To print the value in a form that makes sense (such as the

number of years since the employee's birthday), you must calculate how many years are in 8446 days:

```
AGEYEARS=AGE/365.25;
```

Table 15.4 SAS Date, Time, and Datetime Informat Descriptions

Informat	Description	Width Range	Default Width
DATEw.	dates of form ddMMMyy	7-32	7
DATETIMEw.d	date-time values	13-40	18
DDMMYYw.	date values	6-32	8
MMDDYYw.	date values	6-32	6
MONYYw.	month and year	5-32	5
MSECw.	TIME MIC values	8	8
PDTIMEw.	packed-decimal time from SMF/RMF records	4	4
RMFDURw.	RMF time interval measurements	4	4
RMFSTAMPw.	time-date field from RMF records	8	8
SMFSTAMPw.	time-date field from SMF records	8	8
TIMEw.d	time values	5-32	8
TODSTAMPw.	8-byte time-of-day stamp	8	8
TUw.	timer units	4	4
YYMMDDw.	date values	6-32	8
YYQw.	year and quarter	4-32	4

Date, Time, and Datetime Informat Descriptions

DATEw. informat: ddMMMyy
range: 7-32
default: 7

The DATE. informat reads SAS date values in the form ddMMMyy, where dd is the day of the month, 01-31; MMM is the first three letters of the month name; and yy or yyyy is the year. Blanks and other special characters can be placed before and after the date, and also between the day, month, and year values.

Example The statements

```
DATA DATES;
   INPUT DAY1 DATE10.;
   CARDS;
1JAN1982
01 JAN 82
1 JAN 82
1-JAN-1982
;
```

read each of these values as SAS date values corresponding to 01JAN82.

DATETIMEw.d informat: date and time
range: 13-40
default: 18

Datetime values are those that include both a date and a time, written as the date first and then the time. The date must be in the SAS date form ddMMMyy, followed by a blank or special character, and the time must be in the time form hh:mm, with an optional part ss.ss representing seconds and decimal fractions of seconds. The value of hh ranges from 00 to 23. You must give both a value for date and a value for time.

Example These fields are read with the DATETIMEw. informat as SAS datetime values corresponding to 10:03:17.2 a.m., December 23, 1976:

```
23DEC76:10:03:17.2
23DEC1976/10 03 17.2
```

DDMMYYw. informat: day-month-year
range: 6-32
default: 8

You can use the DDMMYY. informat to read a SAS date value in ddmmyy form, where dd is the day of the month, mm the month, and yy the year. Blanks can be placed before and after the date. The month, day, and year fields can be separated by blanks or special characters. However, if any of these fields are separated by blanks or special characters, all of them must be separated by blanks or special characters.

Example If you use the informat DDMMYY8., the values below are read as 15OCT82:

```
151082
15/10/82
15 10 82
```

MMDDYYw. informat: month-day-year
range: 6-32
default: 8

The MMDDYY. informat reads a SAS date value in mmddyy form, where mm is the month, dd the day of the month, yy the year. Blanks can be placed before and after the date. The month, day, and year fields can be separated by blanks or special characters. However, if any of these fields are separated by blanks or special characters, all of them must be separated by blanks or special characters.

Example The values below are read as 01JAN81 if you use the informat MMDDYY8.:

```
010181
1/1/81
01 1 81
```

MONYYw. informat: month and year
range: 5-32
default: 5

The MONYYw. informat reads SAS date values in the form MMMyy, where MMM is the first three letters of the month name and yy or yyyy is the year.

A value read with the MONYYw. informat results in a SAS date value corresponding to the first day of the specified month. The value must be in the form MMMyy or MMMyyyy. The MMM and yy or yyyy values can not be separated by blanks.

Example The statement

```
INPUT MONTH MONYY5.;
```

reads the field JUN81 and produces a SAS date value corresponding to

```
01JUN81
```

MSECw. informat: TIME MIC values
range: 8
default: 8

The MSEC. informat reads a TIME MIC value (bit 51 equals one microsecond) as a SAS time value. This informat is also useful for reading the difference between two TIME MIC or STCK values or two TODSTAMP. values, whereas TODSTAMP. provides the datetime informat for a stored TOD clock or TIME STCK value.

Example A value produced by an OS TIME macro instruction with the MIC operand (00 00 EA 04 4E 65 A0 00 in hexadecimal) is stored in a record at offset +8. The statements

```
DATA TIME;
   INFILE TRACE;
   INPUT @1  +8 TIMEMIC MSEC8.;
   FORMAT TIMEMIC TIME11.;
   PUT 'THE TIME IS ' TIMEMIC=;
```

read this field and write

```
THE TIME IS TIMEMIC=17:26:58
```

Operating systems: CMS, OS, VM/PC, and VSE

PDTIMEw. informat: packed decimal time of SMF and RMF records
range: 4
default: 4

The PDTIME. informat reads the packed decimal time (in the form 0hhmmssF) of SMF and RMF records as a SAS time value. If the field is all zeros, it is treated as a missing value.

Operating systems: CMS, OS, VM/PC, and VSE

RMFDURw. informat: RMF time interval measurements
range: 4
default: 4

The RMFDUR. informat reads the duration of RMF measurement intervals in RMF records as SAS time values. The data have the form mmsstttF in packed decimal, where mm is minutes, ss is seconds, ttt is thousandths of seconds, and F is the

sign. If the field does not contain packed decimal data, SAS treats the value as missing.

Operating systems: CMS, OS, VM/PC, and VSE

RMFSTAMPw. informat: time and date fields of RMF records
range: 8
default: 8

The RMFSTAMP informat reads the time and date fields of RMF records as SAS datetime values. The first four bytes are 0hhmmssF in packed decimal; the second four bytes contain 00yydddF in packed decimal.

Example

```
DATA A;
   INPUT @2 TYPE PIB1. @3 DATTIM RMFSTAMP8.;
```

Operating systems: CMS, OS, VM/PC, and VSE

SMFSTAMPw. informat: time-date field of SMF records
range: 8
default: 8

The SMFSTAMP. informat reads the time-date field of SMF records as a SAS datetime value. The first four bytes of this field are the time in hundredths of seconds, in integer binary (full-word fixed point), and the next four bytes are the Julian date in packed decimal.

Example

```
DATA A;
   INPUT @2 TYPE PIB1.
         @3 DATTIM SMFSTAMP8.;
```

Operating systems: CMS, OS, VM/PC, and VSE

TIMEw.d informat: hours, minutes, and seconds
range: 5-32
default: 8

The TIME. informat reads SAS time values in the form hh:mm:ss.ss, where hh is the hour, mm the minute, and ss.ss an optional fractional part representing seconds and hundredths of seconds. If you do not give a value for seconds, SAS assumes a value of 0 seconds.

Example These SAS statements read a value for the variable BEGIN:

```
DATA TIMEDATA;
   INPUT BEGIN TIME8.;
   CARDS;
14:22:25
;
```

TODSTAMP. informat: 8-byte time-of-day stamp
range: 8
default: 8

The TODSTAMP informat converts an 8-byte, time-of-day clock value into a SAS datetime value. The time-of-day clock value is returned by the OS TIME macro with the STCK operand, as well as the STCK System/370 instruction. If the 8 bytes

of input contain all zeros, then a regular SAS missing value is returned. The time-of-day clock value is described in *IBM System/370 Principles of Operation* (IBM SRL form number GA22-7000).

Example These SAS statements read a TOD clock value (93 B2 00 C1 9E 7A 20 00 in hexadecimal) stored at offset +49 in a record.

```
INPUT @50 TIMESTCK TODSTAMP8.;
FORMAT TIMESTCK DATETIME17.;
PUT 'THE TIME IS ' TIMESTCK;
```

The SAS log contains the result of the PUT statement:

```
THE TIME IS 03MAY82:17:26:58
```

Operating systems: CMS, OS, VM/PC, and VSE

TUw. informat: timer units
range: 4
default: 4

The TU. informat reads IBM OS/360 and OS/VS (software) timer units as a SAS time value. A time value in (software) timer units is returned by the OS TIME macro with the TU operand. Since there are exactly 38,400 (software) timer units per second, the low-order bit represents approximately 26.041667 microseconds.

Example A value produced by an OS TIME macro instruction with the TU operand (8F C7 A9 BC in hexadecimal) is stored in a record at offset +208. The statements

```
INPUT @1 +208 TIMETU TU4.;
FORMAT TIMETU TIME11.;
PUT 'THE TIME IS ' TIMETU;
```

read this field and write on the SAS log

```
THE TIME IS 17:26:58
```

Operating systems: CMS, OS, VM/PC, and VSE

YYMMDDw. informat: year-month-day
range: 6-32
default: 8

The YYMMDD. informat can be used to read a SAS date value in yymmdd form, where yy is the year, mm the month, dd the day of the month. Blanks can be placed before and after the date, and the month, day, and year fields can be separated by blanks or special characters. However, if any of these fields are separated by blanks or special characters, all of them must be. The YYMMDD. informat accepts a four-digit year value.

Example 1 The data values in the example below are read as 1JAN76:

```
DATA NEWYEAR;
   INPUT BEG YYMMDD8.;
   CARDS;
760101
76 1 1
76-01-01
76/1/1
19760101
;
```

Example 2 The first value below is read as 01JAN1952; the second is read as 16OCT1884.

```
DATA A;
   INPUT DATE YYMMDD10.;
   CARDS;
19520101
1884/10/16
;
```

YYQq. informat: quarters of year
range: 4-32
default: 4

The YYQw. informat reads SAS date values in the form yyQq, where yy or yyyy is the year, Q is the letter Q, and q is the quarter of the year (1, 2, 3, or 4).

A value read with the YYQq. informat produces a SAS date value corresponding to the first day of the specified quarter. For input, the value must be in the form yyQq or yyyyQq. The year value, the letter Q, and the quarter value can not be separated with blanks.

Example The statement

```
INPUT ENTERED YYQ2.;
```

reads the value 82Q2 and gives ENTERED a value of 1MAR1982.

Table 15.5 SAS Date, Time, and Datetime Format Descriptions

Format	Description	Width Range	Default Width
DATEw.	dates of form ddMMMyy	5-9	7
DATETIMEw.d	date-time values	7-40	16
DDMMYYw.	date values	2-8	8
HHMMw.d	hour and minutes	2-20	5
HOURw.d	hour	2-20	2
MMDDYYw.	date values	2-8	8
MMSSw.d	minutes and seconds	2-20	5
MONYYw.	month and year	5-7	5
TIMEw.d	time values	2-20	8
TODw.	time-of-day	2-20	8
WEEKDATEw.	date values	3-37	29
WORDDATEw.	date values	3-32	18
YYMMDDw.	date values	2-8	8
YYQq.	year and quarter	4-6	4

DATEw. format: ddMMMyy
range: 5-9
default: 7

The DATE. format writes SAS date values in the form ddMMMyy, where dd is the day of the month, 01-31; MMM is the first three letters of the month name; and yy or yyyy is the year.

Example

Format	Value printed
DATE5.	12SEP
DATE6.	12SEP
DATE7.	12SEP79
DATE8.	12SEP79
DATE9.	12SEP1979

DATETIMEw.d format: date and time
range: 7-40
default: 16

Datetime values are those that include both a date and a time, with the date followed by the time. The DATETIME. format prints datetime values as DDMMMYY:hh:mm:ss.s. When the field width is 18, the *d* value can be 1. When the field width is greater than 18, the *d* value can be 2.

Example The statement

```
PUT EVENT DATETIME18.;
```

prints the value

```
23DEC81:10:03:17.2
```

Format	Value Printed
DATETIME7.	12SEP79
DATETIME8.	12SEP79
DATETIME9.	12SEP79
DATETIME10.	12SEP79:03
DATETIME11.	12SEP79:03
DATETIME12.	12SEP79:03
DATETIME13.	12SEP79:03:19
DATETIME14.	12SEP79:03:19
DATETIME15.	12SEP79:03:19
DATETIME16.	12SEP79:03:19:43
DATETIME17.	12SEP79:03:19:43
DATETIME18.	12SEP79:03:19:43
DATETIME18.1	12SEP79:03:19:43:2
DATETIME19.2	12SEP79:03:19:43.22
DATETIME20.2	12SEP79:03:19:43.22
DATETIME21.2	12SEP1979:03:19:43.22
DATETIME22.2	12SEP1979:03:19:43.22

DDMMYYw. format: day-month-year
range: 2-8
default: 8

You can use the DDMMYY. format to write a SAS date value in ddmmyy form, where dd is the day of the month, mm the month, and yy the year.

When the field width is from 2 to 5, as much of the month and day values as possible prints. When the width is 7, the date prints with a two-digit year without slashes, and the value is right-aligned in the output field.

Example The DDMMYY6. format writes the date as ddmmyy: for example, 251282. The DDMMYY8. format writes the date as dd/mm/yy: for example, 25/12/82.

HHMMw.d format: hours and minutes
range: 2-20
default: 5

The HHMM. format prints the hours and minutes of a SAS time value, and is similar to the TIME. format except that seconds do not print. If the optional *d* value is given, decimal fractions of minutes print. The SAS System rounds minutes and hours based on the value of the seconds in the SAS time value.

Example 1 The statements

```
DATA NEW;
   TIME='12:34:56'T;
   PUT TIME HHMM.;
```

print

```
12:35
```

Example 2 The statements

```
DATA NEW;
   TIME ='12:59:56'T;
   PUT TIME HHMM.;
```

print

```
13:00
```

HOURw.d format: hours and decimal fractions of hours
range: 2-20
default: 2

The HOUR. format writes only the hour part of a SAS time value. If the optional *d* value is given, decimal fractions of the hour print. SAS rounds hours based on the value of the minutes in the SAS time value.

Example The statements

```
DATA _NULL_;
   TIME='11:30:00'T;
   PUT TIME HOUR4.1;
```

print

```
11.5
```

MMDDYYw. format: month-day-year
range: 2-8
default: 8

The MMDDYY. format writes a SAS date value in mmddyy form, where mm is the month, dd the day of the month, yy the year.

Example The MMDDYY6. format writes the date as mmddyy: for example, 122582. The MMDDYY8. format writes the date as mm/dd/yy: for example, 12/25/82.

Format	Value Printed
MMDDYY2.	09
MMDDYY3.	09
MMDDYY4.	0912
MMDDYY5.	09/12
MMDDYY6.	091279
MMDDYY7.	091279
MMDDYY8.	09/12/79

MMSSw.d format: minutes and seconds
range: 2-20
default: 5

The MMSS. format converts a SAS time value to the number of minutes and seconds since midnight. If the optional *d* value is specified, fractional seconds print as decimals.

Example The statements

```
DATA _NULL_;
   TIME='1:15:30'T;
   PUT TIME MMSS.;
```

write on the SAS log

```
75:30
```

MONYYw. format: month and year
range: 5-7
default: 5

The MONYYw. format writes SAS date values in the form MMMyy, where MMM is the first three letters of the month name and yy or yyyy is the year.

Example The statement

```
PUT ACQUIRED MONYY7.;
```

prints

```
JUN1981.
```

TIMEw.d format: hours, minutes, and seconds
range: 2-20
default: 8

The TIME. format writes SAS time values in the form hh:mm:ss.ss, where hh is the hour, mm the minute, and ss the seconds, with an optional fractional part representing hundredths of seconds.

Example The statement

```
PUT BEGIN TIME.;
```

prints a value of BEGIN as

```
16:24:43
```

TODw. format: time portion of datetime values
range: 2-20
default: 8

The TOD. format converts a SAS time value to a duration from midnight of the day in a datetime value so that you can print the time portion of datetime values.

Example Suppose you read the datetime value 29APR82:3:24:00 into the variable BDTIME. This value is stored internally as the number of seconds between 1JAN60:00:00 and 29APR82:3:24:00. To print just the time part of the value (from midnight, 29APR82), use the TOD. format.

```
PUT BDTIME TOD7.;
```

prints the line

```
3:24:00
```

WEEKDATEw. format: day of week and date
range: 3-37
default: 29

The WEEKDATE. format writes a date value in the form day-of-week, monthname dd, yyyy. If the value of *w* is too small to write the complete day of the week and the month, SAS abbreviates as needed.

Example The statement

```
PUT BEG WEEKDATE.;
```

writes the line

```
MONDAY, NOVEMBER 27, 1979
```

Format	Value Printed
WEEKDATE3.	WED
WEEKDATE9.	WEDNESDAY
WEEKDATE15.	WED, SEP 12, 79
WEEKDATE17.	WED, SEP 12, 1979
WEEKDATE23.	WEDNESDAY, SEP 12, 1979
WEEKDATE29.	WEDNESDAY, SEPTEMBER 12, 1979

WORDDATEw. format: date with name of month written as word
range: 3-32
default: 18

The WORDDATE. format writes a date value in the form month name dd, yyyy.
Example The statement

```
PUT TERM WORDDATE.;
```

writes the value of TERM as

```
SEPTEMBER 30, 1980
```

Format	Value Printed
WORDDATE3.	SEP
WORDDATE9.	SEPTEMBER
WORDDATE12.	SEP 12, 1979
WORDDATE18.	SEPTEMBER 12, 1979
WORDDATE20.	SEPTEMBER 12, 1979

YYMMDDw. format: year-month-day
range: 2-8
default: 8

You can use the YYMMDD. format to write a SAS date value in yymmdd form, where yy is the year, mm the month, dd the day of the month.
Example The statement

```
PUT DAY YYMMDD2.;
```

writes January 1, 1983 as 83. The statement

```
PUT DAY YYMMDD.;
```

writes March 18, 1982 as 82-03-18.

Format	Value Printed
YYMMDD2.	79
YYMMDD3.	79
YYMMDD4.	7909
YYMMDD5.	79-09
YYMMDD6.	790912
YYMMDD7.	790912
YYMMDD8.	79-09-12

YYQq. format: quarters of year
range: 4-6
default: 4

The YYQq. format writes SAS date values in the form yyQq or yyyyQq where yy or yyyy is the year, Q is the letter Q, and q is the quarter of the year (1, 2, 3, or 4).
Example The statement

```
PUT ACQUIRED YYQ4.
```

prints any value of ACQUIRED between 1OCT82 and 31DEC82 as

```
82Q4
```

USING SAS COLUMN-BINARY (MULTIPUNCH) INFORMATS

The maximum number of data items that can be stored on a card is usually eighty: one item of information in each of the 80 columns of the card.

However, it is possible to store more than eighty items on a card by taking advantage of the fact that each of the eighty columns contains twelve positions that can be punched. If the stored item has a narrow range of possible values, only a few of these twelve positions may be needed to store the item.

For example, consider a variable SEX whose values are 1 for females, 2 for males. Only one position of the twelve available in a column is really needed to store a SEX value. If the position is blank, the value is assumed to be 1; if the position is punched, the value is assumed to be 2. Or perhaps two positions might be used: if the first position is punched, the value is 1; if the second is punched, the value is 2.

Consider another variable RACE whose values are 1, 2, 3, or 4. Four positions are needed to store the value.

Both a SEX and a RACE value could thus be stored in a single column: the SEX value in rows 1 and 2, and the RACE value in rows 3-6. Six rows would be unused.

Data stored in this compressed form are called column-binary data or multipunched data. The advantage of using this approach to store data is that more information can be stored than would be possible by using one column per information item. On the other hand, this method requires special card readers, and difficulties are frequently encountered when accessing the data after they have been stored. Thus, the disadvantages outweigh the advantages. Therefore, storing data in a column-binary form is now out of fashion.

Since many multipunched decks and card-image data sets remain in existence, SAS provides informats to read column-binary data.

Because each card column of column-binary data is expanded to two bytes before reading the fields, the LRECL=160 parameter should be specified in the INFILE statement for the input file. For example, if the column-binary data were in a disk data set described by the IN DD statement, the statement

```
INFILE IN LRECL=160;
```

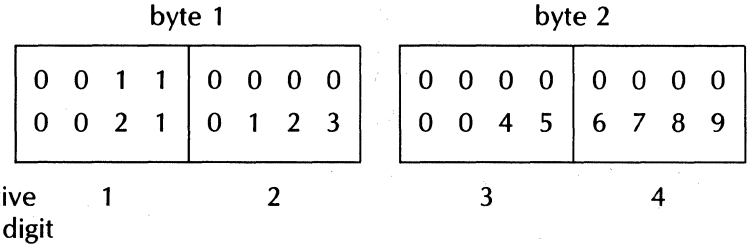
would come before the INPUT statement.

This expansion does **not** affect the column pointer's position. Column addressing with the @ is done as usual since the formats automatically compute the true location on the doubled record. If a value is in column 23, use the pointer control @23 to move the pointer there.

The rows in a card column are numbered from the top as follows:

12
11
(0 or 10)
1
2
3
4
5
6
7
8
9

When column-binary data are transmitted using EBCDIC, the 12-punch bits are divided into two sets of six bits; the six bits are mapped into the end of each 8-bit byte.



Column-Binary (Multipunch) Informat Descriptions

PUNCH.d informat: rows

Using the PUNCH.d informat to read a numeric variable's value results in a value of 1 if row d of the current column is punched; a value of 0 if row d is not punched. The d item can have the values 1 to 9, 10, 11, or 12. After the variable has been read with the PUNCH.d informat, the pointer does **not** advance to the next column.

The example below uses the PUNCH.d informat to read rows 3, 4, 5 in the first column of the card, and rows 11 and 0 from the second column of the card.

```
DATA PUNCH;
   INFILE IN LRECL=160;
   INPUT X3 PUNCH.3 X4 PUNCH.4 X5 PUNCH.5 @2 X11 PUNCH.11
         X0 PUNCH.10;
```

The values of the variables X3, X4, X5, X11, and X0 will be either zeros or ones, depending on whether the corresponding rows are punched on the input card.

Operating systems: CMS, OS, VM/PC, and VSE

ROWw.d informat: columns

The ROW. informat reads a column-binary field down a card column. The w item, which can range from 0 to 12, defines the row where the field begins; the *d* value gives its length in rows. If the *d* value is omitted, it is assumed to be 1. The maximum *d* value is 25. The value assigned the variable being read, which must be a numeric variable, is the relative position of the punch in the field.

For example, suppose that the field to be read begins at row 2 in column 10 and continues through row 5, for a length of four rows. The INPUT statement to read this field would be

```
INPUT @10 X ROW2.4;
```

If row 2 is punched, X's value is 1, since the punch is in the first row of the field. If row 4 is punched, X's value is 3.

The specified field should have no more than one punch. If the field has more than one punch, the variable is assigned a missing value and the automatic variable _ERROR_ is set to 1. If the field has no punches, the variable is also assigned a missing value.

Here is another example. Consider this INPUT statement:

```
INPUT @25 AGE ROW3.3 RACE ROW6.4;
```

Two variables are coded in column 25 of the card. The AGE field begins in row 3 and continues through row 5; the RACE field begins in row 6 and continues through row 9. The values of the AGE and RACE variables are as follows:

RACE: missing if none of the rows 6-9 is punched
missing, _ERROR_=1 if more than one punch in rows 6-9
1 if row 6 is punched
2 if row 7 is punched
3 if row 8 is punched
4 if row 9 is punched

AGE: missing if none of the rows 3-5 is punched
missing, _ERROR_=1 if more than one punch in rows 3-5
1 if row 3 is punched
2 if row 4 is punched
3 if row 5 is punched

Fields read with the ROW. informat may extend to the next column. The scan continues with row 12 of the new column and proceeds down the column. After a field has been read with the ROW. informat, the pointer is set to the row after the last one scanned.

For example, consider this INPUT statement:

```
INPUT @34 X ROW3.3 Y ROW6.4 Z ROW12.15 @37 W $CB1.;
```

X is read from rows 3-5 of column 34; Y is read from rows 6-9 of column 34. Z is read from rows 12-9 of column 35 and rows 12-0 of column 36. W is read from column 37. However, if the @37 pointer control had not been specified, W would have been read from column 36. Use the @ pointer control often to insure a correct pointer position.

Operating systems: CMS, OS, VM/PC, and VSE

$CBw. informat: standard character values

Use the $CB. informat to read standard character data from column-binary files. The column binary code is first translated to standard EBCDIC characters. If the punch combinations are not valid EBCDIC punch codes, the SAS System returns blanks and sets the automatic variable _ERROR_ to 1.

Operating systems: CMS, OS, VM/PC, and VSE

CBw.d informat: standard numeric values

Use the CB. informat to read standard numeric values from column-binary files. The column binary code is first translated to EBCDIC before the number is read from the field. As with the $CB. informat, an invalid punch code results in a missing value and an _ERROR_ value of 1.

Operating systems: CMS, OS, VM/PC, and VSE

Alternative methods Sometimes it may be more convenient to read the 160 bytes of information as if they made up two 80-byte records. In this case, treat

information in columns 41-80 of the record as if it were in columns 1-40 of a second record.

For example, these two groups of statements have the same effect:

```
DATA NEW;
   INFILE COLBIN LRECL=160;
   INPUT @62 X CB2.;
DATA NEW;
   INFILE COLBIN LRECL=80;
   INPUT #2 @22 X CB2.;
```

NOTES

1. **AOS/VS, PRIMOS, and VMS**: The length specification can be either a length variable or an unsigned integer.

   CMS, OS, VM/PC, and VSE: The length must be specified by a length variable.

2. **AOS/VS, PRIMOS, and VMS**: To store formats use the LIBRARY= option in PROC FORMAT to write the formats to a permanent file. Then use a LIBNAME statement identifying the directory containing the formats in later jobs that use those formats.

   CMS, OS, VM/PC, and VSE: To store formats, first create a permanent library (a partitioned data set under OS, a TXTLIB library under CMS and VM/PC, and an existing library or a relocatable library and a core image library under VSE). Use the LIBRARY= option in PROC FORMAT to write the formats to that library. Then include a line identifying the library in the job control language for later jobs that use those formats. (See the "FORMAT" chapter for instructions on storing formats under OS, the *SAS CMS Companion* for instructions under CMS, and the *SAS VSE Companion* for instructions under VSE.)

3. Under OS you can copy the format library to a tape with PROC PDSCOPY.

4. **ASO/VS, PRIMOS, and VMS**: Numbers greater than 999,999,999 print as LARGE_NUMBER.

   CMS, OS, VM/PC, and VSE: Numbers greater than 99,999,999 print as LARGE_NUMBER.

5. **AOS/VS, PRIMOS, and VMS:** Numbers greater than 999,999,999 print as LARGE_NUMBER.

   CMS, OS, VM/PC, and VSE: Numbers greater than 99,999,999 print as LARGE_NUMBER.

6. **AOS/VS, PRIMOS, and VMS**: The length specification can be either a length variable or an unsigned integer.

   CMS, OS, VM/PC, and VSE: The length must be specified by a length variable.

Chapter 16

Missing Values

Introduction

Most collections of data include missing values. For example, if a respondent in a survey fails to answer a given question, an analyst may classify that data item as missing. Sales figures for various products may include a missing value for last year's sales of a product introduced this year. In addition, errors in entering data can produce unidentifiable values.

Values like these can be recognized as missing at the time the SAS System reads them. Other missing values can be produced during the DATA step: you can use program statements to assign missing values to variables, and the SAS System can automatically generate missing values for variables when certain conditions arise.

Missing Values in Input Data Lines

Blanks for missing values You can represent missing values on data lines for numeric and character variables with either blanks or single periods. If you are not using list input, the easiest way to indicate a missing value is simply to leave blank the columns that the value would occupy if it were not missing. When SAS reads the data line, it sets the value to missing.

For example, say your data values are pre- and post-test scores for students. If Susan is absent the day of the post-test, you simply leave Susan's POSTTEST field blank on the input lines:

```
DATA TESTSCOR;
   INPUT NAME $ 1-6 PRETEST 8-9 POSTTEST 11-12;
   CARDS;
ANN    92 96
SUSAN 84
BILL   81 95
more data lines
;
```

Periods for missing values You can also use a single period (.) to represent a missing value. When SAS sees a single period in the columns corresponding to a variable value, it sets the value to missing. (To read a single period as the value of a character variable without having SAS consider the value missing, use the $CHAR. informat.)

If you are using list input, you **must** use periods rather than blanks to represent missing values. With list input, SAS begins reading the next data value at the next non-blank column, so you must use the period to prevent SAS from reading the value of the following variable in place of the value that is missing. (See the description of the INPUT statement in "Statements Used in the DATA Step" for more information about list input.)

For example, say you are using list input to read the test scores in the example above. Use a single period to represent Susan's POSTTEST value:

```
DATA TESTSCOR;
   INPUT NAME $ PRETEST POSTTEST;
   CARDS;
ANN   92 96
SUSAN 84 .
BILL  81 95
more data lines
;
```

Other ways to represent missing values You can also use 9s, 99s, or other values clearly out of the expected range to represent missing values. Although SAS does not automatically read values like these as missing values, you can use program statements to convert them to missing.

For example, say you are reading values of the variable SCORE. The values range from 0 to 7; a value of 9 indicates a missing value. You can use this SAS statement to convert all values of 9 in SCORE to a SAS missing value:

```
IF SCORE=9 THEN SCORE=.;
```

Special missing values At times you may want to differentiate among classes of missing values. For example, in a survey one missing value may mean that the respondent refused to answer a given question, whereas another missing value may mean that the respondent gave an invalid answer.

For numeric variables only, you can designate up to twenty-seven special missing values, the uppercase letters A through Z and the special character underscore (_), when you want to differentiate among missing values. In the survey example you could use the uppercase letter R to indicate that the person refused to respond to a question and the uppercase letter X to indicate that the response is invalid.

See **Working with Missing Values** later in this chapter for information on using special missing values to cause SAS to distinguish between different classes of missing data values.

MISSING statement When your data lines contain characters in numeric fields that you want SAS to interpret as special missing values, you **must** include a MISSING statement. (See a description of the MISSING statement in "Statements Used in the DATA Step.") For example, in the survey above, you might create a data set like this one:

```
DATA SURVEY;
   MISSING R X;
   INPUT ID QUESTN1;
   CARDS;
8401 2
8402 R
8403 1
8404 X
8405 2
more data lines
;
```

in which the letters R and X in the MISSING statement indicate that values of R and X in the input data lines are to be considered special missing values rather than invalid numeric data values.

Illegal characters in input data In addition to blanks or periods and special missing values, you may find that numeric values in your data lines contain illegal characters such as letters or underscores not identified in a MISSING statement, other special characters, or embedded blanks. When SAS encounters these values, it prints a warning message and sets each invalid data value to missing (.).

You can use the SAS option INVALIDDATA to cause SAS to assign a special missing value instead of (.). For example, suppose that you want any invalid data value set to the special missing value A. Use this statement before the DATA step that reads the data lines:

```
OPTIONS INVALIDDATA=A;
```

The INVALIDDATA option does not affect blanks, periods, or values that are specified in a MISSING statement.

Working with Missing Values

Missing initial values At the beginning of each execution of the DATA step, SAS sets the value of each variable you create in the DATA step with an INPUT or assignment statement to missing (except for variables named in a RETAIN statement or created in a SUM statement). SAS then replaces the missing values as it encounters values you assign to the variables. Thus, if you use program statements to create new variables, their values in each observation are missing until you assign the values in an assignment statement.

For example, consider these statements:

```
DATA NEW;
   INPUT X;
   IF X=1 THEN Y=2;
   CARDS;
```

```
4
1
3
1
;
```

When X equals 1, Y's value is set to 2. But since there are no assignment statements that set Y's value when X is not equal to 1, Y remains missing for those observations where X does not equal 1.

When variables are read with a SET, MERGE, or UPDATE statement, SAS sets the values to missing only before the first execution of the SET, MERGE, or UPDATE statement for a given data set (or before the first execution of the statement for each BY group, if a BY statement is present). Thereafter, the variables retain their values until new values become available (for example, through an assignment statement or through the next execution of the SET, MERGE, or UPDATE statement).

When a data set in a match-merge operation (with a BY statement) exhausts its observations, variables it contributes to the output data set retain their values as described above. That is, as long as the BY value in effect when the data set exhausted its observations does not change, the variables it contributes to the output data set retain their values from the final observation. When the BY value changes, the variables are set to missing and remain missing because the data set contains no more observations to provide replacement values. When a data set in a one-to-one merge operation (without a BY statement) exhausts its observations, variables it contributes to the output data set are set to missing and remain missing. See the MERGE statement in "Statements Used in the DATA Step" for examples; see "Introduction to the DATA Step" for more information on the program data vector and how variable values are retained.

Magnitudes of missing values Within the SAS System, a missing value for a numeric variable is smaller than all numbers: if you sort your data set by a numeric variable, observations with missing values for that variable appear first in the sorted data set. For numeric variables, you can compare the special missing values described in **Special Missing Values** above with numbers and with each other as shown in this table. From smallest to largest, the order of magnitude for SAS missing values for numeric variables is

```
_
.
A
B
↓
Z
numbers
```

Missing values of character variables are smaller than any printable character value. However, some usually unprintable characters (for example, machine carriage-control characters, real or binary numeric data that have been read in error as character data) have values less than the blank. Thus, when you sort a data set by a character variable, observations with missing (blank) values of the sorting variable may not appear first in the sorted data set, but they always appear before observations in which values of the sorting variable contain only printable characters.

Specifying missing constants When your input data lines contain special missing values, enter the values without periods in front of them. However, when you use a special missing value in an expression or in an assignment statement, put

a period before the letter or underscore so that SAS will identify it as a missing value instead of a variable name.

For example, suppose that you are checking the ages that people report for themselves in a survey by subtracting their date of birth from the date of the interview and comparing that with their answer to the question on age. If the subtraction gives a different age than they report, you assign a value of .D (for discrepancy) to AGE:

```
DATA SURVEY;
   INPUT SURVDATE BIRTHDAT AGE;
   IF (SURVDATE-BIRTHDAT)¬=AGE THEN AGE=.D;
   CARDS;
data lines
;
```

When SAS prints a special missing value, it prints only the letter or underscore.

Missing values in comparison operations To check for missing values in your data, you can use statements like the following:

```
IF XXX=. THEN DO;
```

for numeric variables, or

```
IF XXX=' ' THEN DO;
```

where ' ' is a blank literal for character variables. In each case, SAS checks to see if XXX's value in the current observation is equal to the missing value specified. If it is, SAS executes the DO group.

Note that, for numeric variables, the first statement checks only for missing values represented by a period (.); it does not check for special missing values such as A or _. If your data contain special missing values, you can check for all missing values of a variable with the following statement:

```
IF XXX<=.Z THEN DO;
```

Since Z is the largest missing value, if any missing values for XXX are present, SAS executes the DO group. To produce a data set containing no observations that have missing values for XXX, use

```
IF XXX>.Z;
```

You can set values to missing within your DATA step with program statements. For example, the statement

```
IF AGE<0 THEN AGE=.;
```

sets the value of AGE to missing if AGE has a value less than 0. Note that if you already have special missing values for AGE, you are resetting them to (.). To avoid resetting them use:

```
IF .Z<AGE<0 THEN AGE=.;
```

Missing values in logical operations Missing values and zero have a value of "false" when you use them with logical operators such as AND or OR. All other values have a value of "true." (See "SAS Expressions" for more information on logical operators.)

Printing missing values SAS prints a period (.) for a missing value of a numeric variable; however, if you have special missing values for numeric variables, it prints the letter or the underscore. For character variables, SAS prints a series of blanks equal to the length of the variable.

You can ask SAS to substitute another character for numeric missing values if you do not want a period (.) printed; however, SAS continues to store the number as (.). Use the SAS option MISSING to define the character you want. For example, to print a blank instead of (.) for numeric missing values, use this statement before the step that prints the values:

```
OPTIONS MISSING=' ';
```

The MISSING option does not affect special missing values: for example, the value .A appears as A in printed output even if you specify another value with the MISSING option. To change special missing values, you must use program statements.

Missing Values Generated by the SAS System

In addition to the missing values that are present in your data and that you assign in program statements, the SAS System can assign missing values to protect you from problems arising in three common computing situations: using missing values in calculations, performing illegal operations, and converting character values to numeric ones when the character variable contains non-numeric information.

Missing values used in arithmetic calculations If you use a missing value in an arithmetic calculation, SAS sets the result of that calculation to missing. Then, if you use that result in another calculation, the next result is also missing. This action is called *propagation of missing values*. Propagation of missing values is important because the SAS System continues working at the same time that it lets you know, via warning messages, which arithmetic expressions have missing values and at what point it created them.

If you do not want missing values to propagate in your arithmetic expressions, you can use the sample statistic functions described in "SAS Functions" to omit missing values from the computations. For example, consider the DATA step below:

```
DATA TEST;
   X=.;
   Y=5;
   A=X+Y;
   B=SUM(X,Y);
RUN;
```

X's value is missing; Y's value is 5. Adding X and Y together in an expression produces a missing result, so A's value is missing. However, using the SUM function to add X and Y produces the value 5 since the SUM function ignores missing values.

Illegal operations If you try to perform an illegal operation (for example, dividing by zero or taking the logarithm of zero), SAS prints a warning message and assigns a missing value to the result.

Illegal character-to-numeric conversions The SAS System automatically converts character values to numeric values if a character variable is used in an arithmetic expression. If a character value contains non-numeric information and SAS tries to convert it to a numeric value, SAS prints an error message and sets the result of the conversion to missing. (For more information about character-to-numeric conversion of data values, see "SAS Expressions.")

Usage Note: Summary

When you are reading data, use the MISSING statement to indicate that numeric fields in your data contain the characters specified in the MISSING statement and that SAS is to read those characters as special missing values. SAS interprets all invalid characters not identified in a MISSING statement as (.). Use the INVALIDDATA option to cause SAS to read all invalid characters in numeric fields (except blanks, periods, and characters identified in a MISSING statement) as the special missing value you choose rather than as (.). When you are printing data, use the SAS option MISSING to print missing values of numeric variables as the character you specify rather than as (.). The MISSING option does not affect character variables or special missing values of numeric variables.

Chapter 17

SAS® Files

INTRODUCTION

To analyze and process information with most SAS procedures, you must first put the information in a *SAS file*. The SAS System can create and process several kinds of SAS files for specialized purposes. When you store a number of SAS files in the same place, you have a *SAS data library*. The first three sections of this chapter describe the kinds of SAS files, how they are created and named, and how they are managed in a SAS data library. The remaining sections of the chapter focus on the *SAS data set*, the most frequently used of the SAS files.

Important Note: in many computing environments the terms *file* and *data set* are used interchangeably. However, this discussion does not equate *SAS file* with *SAS data set*. In this context, a *SAS data set* is a particular kind of *SAS file*. In other words, all SAS data sets are SAS files, but not all SAS files are SAS data sets.

SAS FILES

The SAS System supports a variety of specially structured files called *SAS files*. The special characteristics of SAS files make them more convenient and efficient for SAS programs to use. SAS files are different from *external files*. External files can be processed by other programming languages, as well as by the DATA step's INFILE and FILE statements, the %INCLUDE statement, and several SAS utility procedures. SAS files can be processed only by SAS programs.

The most commonly used SAS file is the *SAS data set*, which contains *observations* for which *variable values* have been recorded. SAS data sets are used with DATA, SET, MERGE, and UPDATE statements and with most SAS procedures.

The other kinds of SAS files are

- catalogs (containing screens for SAS/FSP and SAS/AF software, user function-key profiles, PROC FSCALC spreadsheets, SAS/GRAPH templates, and so on). See the *SAS/FSP User's Guide*, the *SAS/GRAPH User's Guide*, and the *SAS/AF User's Guide* for a description of the catalog and its contents.
- graphics catalogs (containing graphs produced by SAS/GRAPH software). Refer to the *SAS/GRAPH User's Guide* for a description of the graphics catalog and its contents.
- user-written formats (output formats created by PROC FORMAT).[1]
- SAS/IML work space (saved work space from the SAS/IML product). The work space file is described in the *SAS/IML User's Guide*.
- models (models generated by the SAS/ETS product). See the *SAS/ETS User's Guide* for a discussion of model files.

The SAS data set, catalog, and format files are used with the base SAS product and other SAS software. The graphics catalog, model, and work space files are used principally by software other than the base product, although they can be

managed by the base product's CONTENTS, COPY, and DATASETS procedures. You might have applications for all types of SAS files, or you might need only one or two types, such as SAS data sets and formats.[2]

SAS files can be temporary (used in only one job or session) or permanent (stored on disk or tape so they can be used repeatedly). The name of a SAS file determines whether it is temporary or permanent, as discussed in the next section.

NAMING SAS FILES

See the discussion of SAS naming conventions in the chapter "Introduction to the SAS Language" for rules governing names used in SAS programs. A SAS file is named at the time it is created. For example, a SAS data set created in a DATA step is given a name in the DATA statement. A SAS file created in a PROC step is usually given a name in the PROC statement or an OUTPUT statement.[3] You can name a SAS file explicitly, or the SAS System can assign a default name.

Once created, a SAS file is accessed in subsequent DATA or PROC steps by specifying its name. You usually specify the name explicitly, although under certain circumstances, SAS can supply a default name. To use an existing SAS file in a DATA step, specify the file's name in a SET, MERGE, or UPDATE statement (for SAS data sets), or in a PUT or FORMAT statement (for a format). To use an existing SAS file in a PROC step, specify its name in the PROC statement (for SAS data sets, catalogs, saved work space, or model files), or in a FORMAT statement (for formats).

The name you use when creating a SAS file determines where the file is stored, and therefore, whether the file is temporary or permanent.

A SAS file's complete name consists of two words separated by a period, for example, FOOD.PRICES. The first word is called the *first-level name* or *libref*; it indicates where the file is stored. The second word, the *second-level name*, identifies the specific file (PRICES). The second-level name is necessary because there are usually other files stored along with a SAS file.

The libref of a SAS file can be from one to eight characters long. Second-level names are also usually one to eight characters long, but some second-level names are limited to seven characters.[4]

The libref A SAS file's libref (short for "SAS data library reference") is a name that is associated with a storage location. *Storage location* means:

- a *directory* (under AOS/VS, PRIMOS, or VMS)
- a *minidisk* (under CMS or VM/PC)
- an OS data set (under OS)
- a VSE system file (under VSE).

When you create a new SAS file, the libref points to the location where the file is to be stored. When you reference an existing SAS file, the libref indicates the location where SAS should look for the file.

For example, when this statement executes:

```
DATA EMPLOY.DEPT1;
```

SAS writes a file called DEPT1 in the storage location referenced by the name EMPLOY. Suppose that another SAS program uses a PROC PRINT step to print the same SAS file:

```
PROC PRINT DATA=EMPLOY.DEPT1;
```

When the PROC statement executes, SAS looks for the SAS file DEPT1 in the location referenced by the name EMPLOY.

The SAS System knows which storage location a libref references because the libref is associated with the location using SAS statements or operating system control language. Associating a libref with a storage location is also called *defining a libref*. In general, a libref must be defined to be used in a SAS job or session (exceptions are noted where appropriate). Once defined, the libref can be used repeatedly in that job or session, and remains associated with the storage location until the job or session ends (or, under some operating systems, until you change the libref).

The method for defining librefs is different for each operating system. Be sure you read the information in note 5 of this chapter to learn how to define librefs in your operating environment.

SAS does not limit the number of librefs that can be defined in one SAS job or session; however, your operating system or installation might have a limit. Installation personnel can tell you if such a restriction exists.

To create a permanent SAS file or read an existing permanent SAS file, you must define and use a libref other than WORK. The SAS System uses WORK as a default libref for files for which no other libref is defined. All SAS files with a libref of WORK are *temporary files* and are erased at the end of the SAS job or session.

Reserved librefs The SAS System reserves a number of names for particular files or kinds of files. You should not use these reserved names as librefs, except as intended. The reserved names are

FT11F001	LIBRARY
FT12F001	SAS*xxxx*
FT13F001	TAPE*xxxx*
FT14F001	USER
FT15F001	WORK
IXTAPE	

There are other SAS-reserved librefs under some operating systems.[6] In addition, your operating system may have reserved certain words that cannot be used as librefs. Refer to documentation for your operating system for information on reserved names.

Names for Permanent SAS Files

To read or write a permanent SAS file, you usually specify both the first- and second-level names in the SAS statement that requests the file (such as a DATA or PROC statement).

For instance, suppose you want to create a permanent SAS data set with the second-level name PRICES. First, you decide where to store the PRICES data set (that is, in which storage location). Next, you define a libref to reference that location; you choose FOOD as the libref (follow the directions for your operating system). Then, you name the SAS data set in a DATA statement:

```
DATA FOOD.PRICES;
```

When the DATA step executes, a SAS data set named PRICES is stored in the location referenced by the libref FOOD. To read FOOD.PRICES in a subsequent DATA or PROC step in the same SAS job or session, specify both names again. For example

```
PROC SORT DATA=FOOD.PRICES;
   BY ITEM;
```

Since the libref FOOD was defined earlier in the job, it does not need to be redefined. However, if you want to read the PRICES data set in a different job or session, you need to include a libref definition again.

The USER= Option The USER= option is an exception to the rule that you must specify both a libref and second-level name to read or write a permanent SAS file.[7] You can use the SAS system option USER= to specify a default libref so that you need only specify a second-level name to access permanent SAS files. The libref specified with USER= must be defined according to the rules for your operating system.[8]

The USER= option is specified in an OPTIONS statement or when you invoke SAS. The default libref is in effect for the duration of your SAS job or session, or until a different default libref is named by a second USER= option. You can specify only a second-level name for any file that is stored or will be stored in the location referenced by the default libref.

Suppose you want to specify FOOD as the default libref. First, associate FOOD with a storage location. Then, the OPTIONS statement can be written:

```
OPTIONS USER=FOOD;
```

As long as you do not respecify USER= in another OPTIONS statement, FOOD is automatically used as the libref any time a one-level SAS file name is used in that job. For example, this DATA step

```
DATA JUNK;
   SET PRICES;
```

creates a new permanent SAS data set called FOOD.JUNK by reading the permanent SAS data set FOOD.PRICES.

Specifying a default libref with USER= does not mean that files from other storage locations cannot be accessed. However, only those files with the default libref can be referenced using one name.

Note that the USER= option overrides the SAS System's default libref, which is WORK. To create temporary SAS files while the USER= option is in effect, specify a two-level name with WORK as the libref. (WORK does not need to be defined as a libref because SAS assumes that WORK refers to temporary storage.)

Names for Temporary SAS Files

To create or read a temporary SAS file (one that exists only for the duration of the job or session), you usually specify only one name, the second-level name of the file. SAS automatically uses WORK for the libref. WORK is the SAS System's default libref, and it indicates that the file is stored temporarily and will be erased at the end of the SAS job or session.

For example, if your DATA statement is

```
DATA SALES;
```

the new SAS data set's complete name is WORK.SALES. Similarly, to read an existing temporary file, specify only the second-level name:

```
SET COUPONS;
```

When the SET statement executes, SAS reads the file WORK.COUPONS.

Remember that you do not need to define the libref WORK; it is defined automatically when you invoke SAS.

The only time that specifying one name results in a permanent file is when you have specified a default libref other than WORK with the USER= SAS system option.[9]

THE SAS DATA LIBRARY

A libref can be the first-level name of more than one SAS file, as long as all of the SAS files are in the same storage location. All SAS files with the same libref are *members* of one *SAS data library.*[10] SAS does not limit the number of members in a SAS data library.

For an example, consider temporary SAS files, which have the libref WORK. All WORK files from a given job or session are members of the WORK SAS data library.

A SAS data library can contain members of more than one type; for example, one library can include SAS data sets, catalogs, and formats.

When multiple SAS files are members of one library, processing those files with the CONTENTS, COPY, and DATASETS procedures is convenient. That is, you can process more than one member at a time simply by referencing the library. Suppose you have two SAS data sets named LIBONE.SCORES and LIBTWO.COURSES. The libref is different for the two files, so you know that they are not members of the same library.[11] To process both files with PROC CONTENTS, you need two PROC steps:

```
PROC CONTENTS DATA=LIBONE.SCORES;
PROC CONTENTS DATA=LIBTWO.COURSES;
```

If you had stored the files as members of one library, for example, MYSASLIB .SCORES and MYSASLIB.COURSES, you could use one PROC step:

```
PROC CONTENTS DATA=MYSASLIB._ALL_;
```

Another advantage of storing multiple SAS files as members of a SAS data library is that they are logically connected by sharing a common name (their libref). This makes management and tracking of your SAS files easier.

Important Note: in some environments, SAS libraries are physical entities defined and managed by the operating system. In other environments, SAS libraries are not physical entities, but are logical tools that allow you to group SAS files in one place and to reference them with one libref. For the most part, the distinction between a physical and a logical SAS data library is apparent only in the control language you need to set up a SAS job. Be sure to see note 10 for information on the nature of the SAS data library in your operating environment.

The WORK Library

WORK is the libref of the default SAS data library, which is a temporary library. The WORK library is necessary for the operation of SAS. In any SAS job, a number of temporary files (WORK files) are used. Some WORK files you create explicitly, such as WORK SAS data sets. Other WORK files are internally generated by the SAS System and you may not be aware of their existence.

Typically, the WORK library is erased automatically at the end of a SAS job. A new WORK library is defined automatically at the beginning of a SAS job.

Duplicate Names

You can assign the same second-level name to more than one SAS file in a SAS data library, as long as the files are of different types. For example, suppose you have a SAS data library with the libref LIB2, and you are creating a SAS data set and a SAS catalog that will be stored in LIB2. You could give both files the same second-level name, for example, ADDRESS. SAS distinguishes the files from each other by checking their type.

We do not recommend that you duplicate names within a library. However, if you do, be careful when using the COPY, CONTENTS, and DATASETS utility procedures so that you know which files are being processed.

SAS FILE MANAGEMENT

The SAS utility procedures available for SAS file management are designed to allow you to work with more than one SAS file at a time, as long as the files belong to the same SAS data library. Therefore, we often discuss file management in terms of SAS data libraries, whether the library is a physical or logical entity, and whether the library contains one or many SAS files.

Storing SAS Data Libraries on Disk or Tape

Under most operating systems,[12] you can choose to store SAS data libraries on either disk or tape. Some considerations that may affect your decision are given here:[13]

- Under some operating systems, only SAS data sets can be stored on tape. Other types of SAS files cannot be stored on tape, which means that a SAS data library on tape can contain only SAS data sets.[14]
- Data stored on on-line disk are always available, and your SAS jobs do not have to wait for a tape or off-line disk to be mounted.
- At some computer installations, only on-line disk storage is possible. Users do not have access to tapes.
- Tape storage is less flexible than disk storage since the file you want to use cannot be accessed directly.
- On-line disk space is usually more expensive than off-line disk space and tapes.
- Tapes are able to accommodate large collections of data for which disk storage may be impractical.

Other considerations specific to your installation may affect your choice of disk or tape. Check with the staff of your computer installation if you are not familiar with using disks and tapes.

SAS data libraries on tape When you store SAS data sets on tape, keep these points in mind:

- At any point in a SAS job, you can access only one of the SAS files on a tape. For example, you cannot read two SAS data sets on the same tape in a single DATA step. However, you can access two or more SAS files on different tapes at the same time, if there are enough tape drives available to your job. You can also access a SAS file on a tape during one DATA or PROC step in the job, then access another SAS file on the tape during a later DATA or PROC step.
- If you write over or replace an existing SAS file on tape, other files after it on the tape are inaccessible.[15]

Tape-format data libraries on disk SAS data libraries can be stored on disk in tape format (sequential) under some operating systems. The libref of a tape-format disk SAS data library must begin with the characters TAPE.[16]

Replacing SAS Files

If you are creating a SAS file that has the same name as an existing file of the same type in your SAS data library, the original file is deleted by default after the

new file is written successfully.[17] However, if errors occur before the new file is finished, the original file is preserved.

You can allow or disallow replacement of a permanent file with the REPLACE/NOREPLACE system option. REPLACE is the default value; if you do not want files to be automatically replaced, specify NOREPLACE as a system option.

In addition, for SAS data sets only, you can override the REPLACE | NOREPLACE system option with the REPLACE= data set option.[18] For example, the NOREPLACE system option is in effect in the SAS job shown below, but it is temporarily overridden for the FOOD.MEAT SAS data set.

```
OPTIONS NOREPLACE;
DATA FOOD.MILK;
    SET FOOD.DAIRY;
    IF PRODUCT=MILK;
DATA FOOD.MEAT (REPLACE=YES);
    SET FOOD.MEAT;
    IF DATE>'1JUL84'D;
```

Documenting the Contents of SAS Data Libraries

PROC CONTENTS gives you complete documentation on the contents of the SAS data library and the files it contains. The type of each SAS file is shown, and for SAS data sets, variable names and labels, the number of observations, and other information can be displayed by PROC CONTENTS. See the CONTENTS procedure for complete information.

Copying SAS Data Libraries

PROC COPY can be used to copy SAS data libraries from disk to disk, disk to tape, tape to disk, and tape to tape. It is especially designed for backups.

Under some operating systems, SAS data libraries should be copied only with the COPY procedure.[19] Files may become unreadable if you use other, non-SAS, utility procedures to copy them. See "The COPY Procedure" for a complete description.

If copying to tape, remember that under some operating systems, only SAS data sets can be stored on tape.[20]

Do not confuse PROC COPY with PROC XCOPY. PROC XCOPY is needed to move SAS data libraries between certain operating systems. See the discussion on transporting SAS files between operating systems later in this chapter and "The XCOPY Procedure" for more information.

Renaming and Deleting SAS Files

You can use PROC DATASETS to rename disk-format SAS files, delete SAS files, and systematically rename a group of functionally related files in the same SAS data library. DATASETS also allows you to rename and relabel variables in SAS data sets. See "The DATASETS Procedure" for more information.

Transporting SAS Files to Other Computers

In many cases, you can move SAS files from one computer to another, if necessary. The method for moving one or more SAS files varies, depending on the kind of file, the operating system on which it is originally stored, and the operating system to which it is being moved.

Under some circumstances, it is possible to move a regular SAS file to a different computer using the DATA step or PROC COPY and disks connected by a communications network, or tape. In other cases, regular or standard SAS files cannot

be moved to another computer; instead, the SAS file must first be rewritten in *transport format*.

SAS data sets can be written in transport format, and SAS catalogs can also be written in transport format. Other kinds of SAS files (graphics catalogs, models, IML workspace, formats) cannot be written in transport-format. A *transport-format SAS library* is a SAS data library containing one or more transport-format SAS files. Note that a SAS data library cannot contain both standard and transport-format SAS files. In this discussion, a *SAS data library* is one containing standard SAS files, and a *transport library* is one containing transport-format SAS files.[21]

A transport-format library can contain only transport-format SAS data sets or transport-format catalogs. A transport library cannot contain both kinds of SAS files. In addition, a transport library can contain only one transport-format SAS catalog.

Transport-format libraries can be written to disk or tape. When written on tape, the libref (first-level name) **must** be IXTAPE under some operating systems.[22]

How to create a transport library Transport-format SAS data sets are created by:

- **PROC COPY** with the EXPORT option (AOS/VS, PRIMOS, and VMS only)
- **PROC XCOPY** with the EXPORT option (CMS, OS, and VSE only)
- a **DATA step** with the TRANSPORT= option (AOS/VS, PRIMOS, and VMS only).

Transport-format SAS catalogs are created by:

- **PROC CPORT** from the SAS/AF software product (CMS, OS, and VSE only).

How to read a transport library To read a transport-format SAS data set, use:

- **PROC COPY** with the IMPORT option (AOS/VS, PRIMOS, and VMS only)
- **PROC XCOPY** with the IMPORT option (CMS, OS, and VSE only)
- a **DATA step** with the TRANSPORT= option (AOS/VS, PRIMOS, and VMS only).

Transport-format SAS catalogs are read by:

- **PROC CIMPORT** from the SAS/AF software product (AOS/VS, PRIMOS, and VMS only).

See the primary documentation for these procedures and options for details and examples of their use. (The CPORT and CIMPORT procedures are documented in the *SAS/AF User's Guide*.)

Under some operating systems,[23] SAS files can also be transported using operating system utility programs, as long as the files are being moved to a computer running the same operating system. You may find this useful for transporting SAS files that cannot be transported with a SAS program or if you want to transport more than one kind of SAS file at a time.

The following tables show how to move SAS files from one computer to another. There is a table for each operating system, with columns for types of SAS files and rows for each operating system to which a file might be moved. For example, if you are a CMS SAS user and you want to know how to move a SAS file to a computer running the AOS/VS operating system, see the table titled **Transporting SAS Files From CMS to Other Computers**.

Each cell in the tables indicates whether the SAS file must be in standard or transport format and what storage medium can be used: tape, disk, or diskette.

If the word disk appears and is followed in parentheses by the word "sequential," it means that only a sequential file is allowed if stored on disk. Where disk storage is valid, it implies that a communications network between the computers must exist.

If the word NO appears in a cell, it means that the file cannot be moved to the specified operating system.

Remember: if the table indicates that transport-format is required, a transport format library must be created on the computer originating the file and read on the computer receiving the file. See the lists in the previous section on how to create and read transport-format files.

Table 17.1 Transporting SAS Files from AOS/VS to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	standard format on tape or disk (sequential). Or use AOS/VS utility	AOS/VS utility	AOS/VS utility
CMS	transport format on tape	NO	NO
OS	transport format on tape	NO	NO
PRIMOS	transport format on tape	NO	NO
VM/PC	NO	NO	NO
VMS	transport format on tape	NO	NO
VSE	transport format on tape	NO	NO

Table 17.2 Transporting SAS Files from CMS to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	transport format on tape	transport format on tape	NO
CMS	standard format on tape or disk	standard format on tape or disk	standard format on tape or disk
OS	standard format on tape	standard format on tape	standard format on tape
PRIMOS	transport format on tape	transport format on tape	NO
VM/PC	standard format on disk	standard format on disk	standard format on disk
VMS	transport format on tape	transport format on tape	NO
VSE	standard format on tape	standard format on tape	standard format on tape

Table 17.3 Transporting SAS Files from OS to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	transport format on tape	transport format on tape	NO
CMS	standard format on tape or disk	standard format on tape or disk	standard format on tape or disk
OS	standard format on tape or disk	standard format on tape or disk	standard format on tape or disk
PRIMOS	transport format on tape	transport format on tape	NO
VM/PC	standard format on disk	standard format on tape	standard format on disk
VMS	transport format on tape	transport format on disk	NO
VSE	standard format on tape	standard format on tape	standard format on tape

Table 17.4 Transporting SAS Files from PRIMOS to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	transport format on tape	NO	NO
CMS	transport format on tape	NO	NO
OS	transport format on tape	NO	NO
PRIMOS	standard format on tape or disk (sequential) or use PRIMOS utility	PRIMOS utility	PRIMOS utility
VM/PC	NO	NO	NO
VMS	transport format on tape	NO	NO
VSE	transport format on tape	NO	NO

Table 17.5 Transporting SAS Files from VM/PC to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	NO	NO	NO
CMS	standard format on disk	standard format on disk	standard format on disk
OS	NO	NO	NO
PRIMOS	NO	NO	NO
VM/PC	standard format on diskette	standard format on diskette	standard format on diskette
VMS	NO	NO	NO
VSE	NO	NO	NO

Table 17.6 Transporting SAS Files from VMS to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	transport format on tape	NO	NO
CMS	transport format on tape	NO	NO
OS	transport format on tape	NO	NO
PRIMOS	transport format on tape	NO	NO
VM/PC	NO	NO	NO
VMS	standard format on tape or disk (sequential) or use VMS utility	VMS utility	VMS utility
VSE	transport format on tape	NO	NO

Table 17.7 Transporting SAS Files from VSE to Other Computers

To:	SAS Data Set	SAS Catalog	Other SAS Files
AOS/VS	transport format on tape	transport format on tape	NO
CMS	standard format on tape	standard format on tape	standard format on tape
OS	standard format on tape	standard format on tape	standard format on tape
PRIMOS	transport format on tape	transport format on tape	NO
VM/PC	NO	NO	NO
VMS	transport format on tape	transport format on tape	NO
VSE	standard format on tape	standard format on tape	standard format on tape

SAS DATA SETS

Definition

SAS data sets are the most frequently used type of SAS file. They are created and read in the DATA step and by SAS procedures. See "Introduction to the DATA Step" and "Statements Used in the DATA Step" for details on creating SAS data sets. Also consult procedure documentation about the types of SAS data sets created by different procedures and for the syntax required to create them. The **Special SAS Data Sets** section in this chapter describes some of the SAS data sets that can be created with SAS procedures.

A *SAS data set* is a computer file of data packaged in a convenient way for SAS software to use. SAS data sets contain *observations* for which *variable values* (data) have been recorded, descriptor information on the variables, and the history of the data set.[24] SAS data sets can be permanent or temporary.

SAS data sets differ from external files in the way they store data and because they automatically maintain descriptive information about the data set. The data are arranged in a rectangular table with the rows of the table representing the observations and the columns representing the variables. The descriptor information is at the beginning of each SAS data set and includes the names and certain attributes of the variables in the data set. The attributes stored for each variable are

- name (from one to eight characters in length)
- type (character or numeric)
- length (the number of bytes used to store a variable)
- position (the location of the variable within an observation)
- informat name (for reading the variable values)
- format name (for printing the values)
- label (a variable descriptor consisting of from one to forty characters).

In some cases,[25] the descriptor section of a SAS data set also contains history information: the date and time of the data set's creation, the name of the job that created it, and the SAS statements used to create it. You can specify how many generations of this history information are stored for a SAS data set with the SAS system option GEN=.

The maximum allowable sizes of a SAS data set and the observations in a data set vary according to operating system.[26]

Whether you intend to store a SAS data set permanently or temporarily, you must have your data in a SAS data set in order to use most SAS procedures. SAS relies on the information in the descriptor section to process variable values correctly.

Further Notes on SAS Data Set Names

Specifying _NULL_ If you want to execute a DATA step but do not want to create a SAS data set, you can specify _NULL_ in the DATA statement, rather than a SAS data set name:

```
DATA _NULL_;
```

For example, if you are writing a report based on the values in a SAS data set, you probably do not want to create another SAS data set containing the same information. Using _NULL_ means that SAS executes the DATA step just as if it were creating a new SAS data set, but does not write any observations. This can be a more efficient use of computer resources.

Omitting a name If you do not specify a SAS data set name or _NULL_ in the DATA statement, a temporary SAS data set is created. The temporary data set's name is WORK.DATA*n*, where *n* is the number of the data set. For example, the first such SAS data set created is WORK.DATA1, the second is WORK.DATA2, and so on.

The _LAST_ data set The SAS System uses a special automatic variable called _LAST_ to keep track of the most recently created SAS data set in a SAS job or session. The value of _LAST_ is null initially, but each time a new SAS data set is created, the value of the _LAST_ variable changes to the name of the newest data set.

When you execute a SAS procedure without specifying a SAS data set, SAS, by default, uses the _LAST_ data set. For example

```
DATA FOOD.DAIRY;
   SET FOOD.PRICES;
   IF DEPT='DAIRY';
PROC PRINT;
```

After the DATA step executes, the value of the _LAST_variable is FOOD.DAIRY. The PROC PRINT statement does not specify a SAS data set name, so SAS uses FOOD.DAIRY (the _LAST_ data set) by default.

The SAS system option _LAST_= allows you to explicitly assign a value to the _LAST_ variable. The value you assign with the _LAST_= option remains the value of the _LAST_ variable until a new SAS data set is created.

You might use the _LAST_ option when, for example, you want to run a number of PROC steps using an existing permanent SAS data set. By using _LAST_=, you avoid the need to specify a SAS data set name in each PROC statement.

Advantages of Permanent SAS Data Sets

There are several advantages to storing your data in a permanent SAS data set rather than leaving them in an external file or recreating a temporary data set in every SAS job:

- You leave reading the data to SAS; you need not be concerned about format, and you do not need to execute INPUT statements each time a SAS data set is used.
- SAS automatically documents the SAS data set, and you can keep track of its contents easily. Using SAS utility procedures, you can always find out which variables the data set contains, their lengths and formats, and other information that often gets lost for undocumented files.
- No data conversion is necessary since data are stored in the form in which SAS uses them, saving computer time.

SAS DATA SET OPTIONS

Data set options are those that appear after SAS data set names. They specify actions that are applicable only to the processing of the SAS data set with which they appear and let you perform such operations as:

- giving a descriptive label to a SAS data set
- specifying variables to be included or dropped in later processing
- selecting only the first or last *n* observations for processing.

SAS data set options apply only to SAS data sets, not to any other type of SAS file.

Data set options are specified in parentheses after the SAS data set name in DATA step or PROC statements, for example

```
DATA NEW (DROP=YEAR);
```

The DROP= option specifies that a variable called YEAR be dropped from the SAS data set called NEW.

To specify two or more options, leave at least one space between them in the parenthesized list. For example

```
DATA NEW(DROP=YEAR FIRSTOBS=101);
```

Some data set options are valid only when a SAS data set is created; for example they can appear in a DATA statement but not in a SET statement. Other options are valid only when an existing data set is being read, as in a SET, MERGE, or UPDATE statement. Some options can be used in both situations.

The data set options are listed below. The accompanying explanation gives the circumstances under which the option can be specified and the operating systems to which it applies.

BLKSIZE=*blocksize*
: specifies the block size used to write the SAS data set. If the block size specified is too small to contain one observation, SAS uses a value large enough to contain one observation. If the value is not four plus a multiple of the logical record length, SAS computes the block size with the formula

  (FLOOR(*blocksize*/*lrecl*)**lrecl*)+4

  where *blocksize* is the value you specified for BLKSIZE=, and *lrecl* is the observation length plus four. BLKSIZE= can only be specified when a SAS data set is created.

  Operating systems: CMS, OS, VM/PC, and VSE

BUFNO=*n*
: specifies the number of input buffers used when accessing SAS data sets. The value specified may be either 1 or 2. See "Introduction to the DATA Step" for a discussion of input buffers.

  Operating systems: OS and VSE

DROP=*variables*
: causes the specified variables to be omitted from the SAS data set that is being created or during the processing of the data set. If the DROP= option appears in a DATA statement and only one data set is being created, DROP= functions exactly as the DROP program statement does. If the DATA statement specifies several data sets, the DROP= option can be used to control which variables appear in which data sets. For example, consider the following statements:

```
DATA HISCHOOL (DROP=COLLNAME COLLCODE) COLLEGE;
   INPUT YRS_EDUC HSNAME $ COLLNAME $ COLLCODE;
   IF YRS_EDUC<=12 THEN OUTPUT HISCHOOL;
   IF YRS_EDUC>12 THEN OUTPUT COLLEGE;
```

  The SAS data set COLLEGE contains all the variables in the INPUT statement; the data set HISCHOOL includes all the variables except COLLNAME and COLLCODE.

The DROP= option is useful with the OUT= option on a PROC statement. For example, you may want to plot residuals produced with the OUTPUT statement of PROC GLM against the dependent variable. The procedure normally includes all variables from the original SAS data set in the output SAS data set. The following statements produce an output data set containing only the residuals and the dependent variable values and ask PROC PLOT to plot those values against each other:

```
PROC GLM DATA=EXP;
   MODEL Y=X1-X5;
   OUTPUT  OUT=RESID(DROP=X1-X5)
      RESIDUAL=RESID Y;
PROC PLOT;
   PLOT Y*RESID Y;
```

The DROP= option can also appear when an existing SAS data set is being processed. The listed variables are not available to SAS during the processing. This could be useful if, for example, you want to update only some of the variables of a data set. Variables that are not to be updated could be excluded from the update operation with a DROP= option:

```
DATA NEW;
   UPDATE OLD (DROP=PAYCODE) UPS;
   BY SSN;
```

Operating systems: All

FILECLOSE=*position*
specifies the volume positioning to be performed when a SAS data library on tape is closed. The values that can be specified are REREAD, REWIND, LEAVE, DISP, and FREE.[27] The FILECLOSE= option overrides the SAS system option TAPECLOSE=.

Specify FILECLOSE=REREAD if you intend to use that tape data library again in the same SAS job. SAS leaves the tape volume positioned at the beginning of the data library. REREAD overrides a FREE=CLOSE specification in the job control.

Specify FILECLOSE=REWIND if you do not want to use the tape volume again in the same job or if you want to use the first data library (file) on the tape volume. SAS rewinds the tape volume to the beginning. A FREE=CLOSE specification in the job control overrides the REWIND specification.

Specify FILECLOSE=LEAVE if you are not using the tape data library again in that job, but you are creating or accessing a subsequent tape file on the same tape volume. SAS leaves the tape positioned at the end of the data library. LEAVE overrides a FREE=CLOSE specification in the job control.

FILECLOSE=DISP[28] causes the volume to be positioned as determined by the operating system, according to specifications in the job control.

FILECLOSE=FREE[29] causes the data library to be dynamically deallocated under MVS operating systems so that thereafter it is not possible to access it. FREE is equivalent to the job control specification FREE=CLOSE. If you try to access the data library, SAS issues an error message that the libref is not defined. The FREE specification is useful for freeing the data library allocation during a

long job, so that other jobs and users can allocate the library, tape volume, tape drive, or unit.

Operating systems: OS and VSE

FILEDISP=NEW
FILEDISP=OLD
specifies the initial disposition (status) for a SAS data library on tape. This option is used only when a SAS data set is created.[30]

Operating systems: CMS, OS, and VSE

FIRSTOBS=*n*
causes processing to begin with the *n*th observation. The *n* value must be a positive integer. For example, the statement

```
PROC PRINT DATA=STUDY (FIRSTOBS=20);
```

results in printing observations beginning with number 20. This option is valid only when reading an existing SAS data set.

Operating systems: All

GEN=*n*
specifies the number of generations (from 0 to 1000) of historical information that SAS keeps for a SAS data set. This information can be printed with the CONTENTS procedure. If GEN=2 is specified, for example, SAS keeps information on the creation of the current data set and on the previous generation of SAS data sets from which the current one was built. GEN= is valid for new or existing data sets.

Operating systems: CMS, OS, VM/PC, and VSE

IN=*variable*
names a new variable in a SET, MERGE, or UPDATE statement that contains values indicating the data set from which an observation comes. The variable's value is 1 if values in the current observation were taken from that data set, and it is 0 otherwise. The IN= option is specified in parentheses after a SAS data set name in the SET, MERGE, or UPDATE statement, for example,

```
MERGE FOOD.DAIRY(IN=INDAIRY) FOOD.MEAT;
```

Values of IN=variables are available to program statements during the DATA step, but the variables are not included in the SAS data set being created.

Operating systems: All

KEEP=*variables*
causes only the listed variables to be retained for processing or output to the SAS data set. If KEEP= appears when a SAS data set is created, only the listed variables appear in the new data set. KEEP= is useful when several data sets are created with one DATA statement: it can specify which variables are to be included in which data sets.

If the KEEP= option is used when an existing data set is read, only the variables listed are available to SAS during processing. The variables not listed are still in the data set, however.

Operating systems: All

LABEL=*label*

specifies a label for the SAS data set, which is stored with the data set and printed whenever the CONTENTS procedure is used to print the data set's contents. The label consists of up to 40 characters and should be enclosed in quotes. If the label characters include single quotes, write them as two single quotes and enclose the entire label in single quotes. For example

```
DATA W2(LABEL='1976 W2 INFO, HOURLY');
DATA NEW(LABEL='DAVE''S LIST');
DATA SALES(LABEL='SALES FOR MAY(NE)');
```

The LABEL= option is used only when a data set is created.

Operating systems: All

OBS=*n*

specifies the last observation of the SAS data set that will be processed. (Note OBS= does not specify how many observations should be processed.) The *n* value must be a positive integer. This option is valid only when reading an existing data set.

Operating systems: All

PROTECT=*password*

specifies a password that protects a SAS data set from alteration and deletion. The PROTECT= option can only be used to assign a password when a data set is created or in the DATASETS procedure. If the PROTECT= option is in effect, another data set of the same name cannot be created unless the password is given with the PROTECT= option, nor can the data set be deleted or modified without the option.

For example, suppose an instructor creates a SAS data set using these SAS statements:

```
DATA CLASS3.EXAMS (PROTECT=ST361);
   INPUT SSN  SCORE1 2. SCORE2 2. SCORE3 2. GRADE $2.;
```

The data set CLASS3.EXAMS can be used in PROC statements and in SET, MERGE, and UPDATE statements (that is, it can be read). It cannot be used in DATA statements or with any procedure that writes or updates a data set with the same name (for example, PROC EDITOR, PROC SORT, PROC RANK, or PROC APPEND) unless PROTECT=ST361 appears also (that is, it cannot be altered or deleted).

Operating systems: CMS, OS, VM/PC, and VSE

READ=*password*

specifies a password that protects the SAS data set from being read unless that same password is given. The password must be a valid SAS name.

For example, if the statement

```
DATA CORP.SALARY(READ=EXEC);
```

is executed, the SAS data set CORP.SALARY can only be used if READ=EXEC is specified. The statement

```
PROC MEANS DATA=CORP.SALARY;
```

would therefore fail, but the statement:

```
PROC MEANS DATA=CORP.SALARY(READ=EXEC);
```

would be executed.

The READ= option can only be used to assign a password when the SAS data set is created or with PROC DATASETS. For example, if you did not assign CORP.SALARY a password when it was created, you could use these statements to assign one:

```
PROC DATASETS DDNAME=CORP;
   MODIFY SALARY (READ=EXEC);
```

A data set protected with a READ= password but not a PROTECT= password is not fully protected from alteration and deletion.

Operating systems: CMS, OS, VM/PC, and VSE

RENAME=(*oldname=newname...*)
: changes the name of a variable. If RENAME= is specified when a data set is created, the new name is permanent. If RENAME= is specified at any other time, the new name exists only for the duration of the procedure. For example, the statements

```
DATA NEW (RENAME=(X=KEYS));
   SET OLD;
```

create the SAS data set NEW. NEW contains the same variable values as data set OLD, however, the variable named X in data set OLD is named KEYS in data set NEW.

Several variables can be renamed with one RENAME option, for example

```
DATA NEW (RENAME=(X=KEYS Y=LOCKS);
```

If RENAME= is used and either DROP= or KEEP= is also used, DROP= and KEEP= are applied before RENAME=. Thus, use the *oldname* in the KEEP= or DROP= option.

You cannot use an abbreviated variable list (for example, X1-X10) with the RENAME= option.

Operating systems: All

REPLACE=YES
REPLACE=NO
: is used to override the REPLACE/NOREPLACE system option allowing replacement of permanent SAS data sets.

Operating systems: CMS, OS, VM/PC, and VSE

TRANSPORT=NO
TRANSPORT=YES
: specifies whether or not the SAS data set is in transport format. TRANSPORT=NO, the default, means that the SAS data set is in standard SAS form. If TRANSPORT=YES, it means that the SAS data set has been changed to transport format to be moved to another operating system or machine.

You can use the TRANSPORT= data set option to create or read a transport-format SAS data set in the DATA step or in any procedure where data set options are allowed. Two limitations are

- only one transport SAS data set can be accessed in a DATA step
- transport data sets cannot be randomly accessed (such as with the POINT= option on a SET statement).

For example, to read a transport data set stored on tape, you can use the statements:

```
DATA NEW;
   SET IXTAPE.OLD(TRANSPORT=YES);
```

The example creates a standard format SAS data set called NEW from a transport data set called IXTAPE.OLD. (The libref IXTAPE is reserved for transport data libraries on tape. Any libref can be used for a transport data library on disk.)

When you use the TRANSPORT= data set option to create a SAS data set in transport format, as in

```
DATA IXTAPE.AR233(TRANSPORT=YES);
   SET STOCK;
```

you create a transport data library with only one member, AR233.

You cannot use the DATA step with the TRANSPORT= option to write a multi-member transport library, but you can read a transport-format data set from a multi-member transport library. For more information, see the section on transporting SAS files between operating systems. Also see the COPY procedure.

Operating systems: AOS/VS, PRIMOS, and VMS

TYPE=DATA
TYPE=CORR
TYPE=COV
TYPE=SSCP
TYPE=EST
TYPE=FACTOR
TYPE=DISCAL

specifies a special SAS data set type for input data. The TYPE= option's primary use is in the CANCORR, CANDISC, PRINCOMP, VARCLUS, DISCRIM, FACTOR, SCORE, and REG procedures, which accept data in specially structured SAS data sets. These special data sets are usually created by earlier runs of one of these procedures. The TYPE= option need not be used if the special data set was created by a SAS procedure. It would be used, for example, if correlation values had been produced by a non-SAS program. In this case, the values would have to be put into a SAS data set with the proper format. Here is an example of the TYPE= option's use in such a case:

```
PROC FACTOR DATA=OLD(TYPE=CORR);
```

The TYPE=DATA data set is an ordinary SAS data set. Each of the other types of SAS data sets is described below in **Special SAS Data Sets**.

Operating systems: All

SPECIAL SAS DATA SETS

In addition to standard SAS data sets and transport-format SAS data sets, there are several specially structured SAS data sets that are used by some SAS statistical procedures. These SAS data sets are in standard format, but they contain special variables and observations. These special data sets are usually created by SAS statistical procedures, but you can also use a DATA step to create a special SAS data set in the proper format. If created by a DATA step, you use the TYPE= data set option to indicate the data set's type to SAS.

TYPE=CORR Data Sets

A TYPE=CORR data set contains a correlation matrix along with the variable means, standard deviations, the number of observations in the original SAS data set from which the correlation matrix was computed, and possibly other statistics (depending on which procedure created the SAS data set).

Using PROC CORR with an output data set specification automatically produces a TYPE=CORR data set. You can also create a TYPE=CORR data set from input data that contain a correlation matrix (see the example below). In this case, TYPE=CORR must be specified as a data set option.

TYPE=CORR data sets can be used as input for PROC FACTOR, PROC REG, and other procedures.

Variables in a TYPE=CORR data set When a BY statement is used with PROC CORR, the BY variable(s) appears first in the data set. Next come two special character variables, each eight characters long. The first is named _TYPE_, and its values identify the type of each observation in the TYPE=CORR data set (MEAN, STD, N, CORR). The second special variable is named _NAME_, and its values identify the variable with which a given row of the correlation matrix is associated. The variables from the original data set that were analyzed by PROC CORR come next.

Observations in a TYPE=CORR data set For the first observation, which contains the variable mean, the _TYPE_ variable's value is 'MEAN'; for the second observation, containing standard deviations, _TYPE_'s value is 'STD'; for the third observation, containing the number of observations, _TYPE_'s value is 'N'. The _NAME_ variable's value is blank for these first three observations.

The first three observations are produced when PROC CORR creates the TYPE=CORR data set. However, if you create the TYPE=CORR data set, the data set need not contain these three observations. Any procedure that uses the data set uses 0 for all the variable means, 1 for all the standard deviations, and 100 for the number of observations, with the exception of CANCORR, CANDISC, FACTOR, PRINCOMP, RSQUARE, STEPDISC, and VARCLUS, which use 10,000 for the number of observations.

Following the first three observations are the observations containing the correlation matrix; one for each row of the matrix. _TYPE_'s value for each of these observations is 'CORR'. _NAME_'s value for each observation is the variable name associated with that observation (row).

A TYPE=CORR data set: example 1 See **Output 17.1** for an example of a TYPE=CORR data set containing a 2-variable correlation matrix. In the output, 12.2 and −4.5 are the means of X and Y; 3.2 and 1.1 are the standard deviations of X and Y; 5 is the number of observations containing X and Y; and .7 is the correlation between X and Y.

Output 17.1 A TYPE=CORR SAS Data Set with Two-variable Matrix

OBS	_TYPE_	_NAME_	X	Y
1	MEAN		12.2	-4.5
2	STD		3.2	1.1
3	N		5.0	5.0
4	CORR	X	1.0	0.7
5	CORR	Y	0.7	1.0

Using BY variables: example 2 **Output 17.2** shows an example of a TYPE=CORR data set created with PROC CORR and a BY statement:

```
PROC CORR DATA=MEASURE OUTP=CORMAT;
   BY SEX;
   VAR A B C;
PROC PRINT;
```

Output 17.2 A TYPE=CORR SAS Data Set with BY Variables

OBS	SEX	_TYPE_	_NAME_	A	B	C
1	F	MEAN		14.7	29.6	9.6
2	F	STD		1.2	3.1	0.7
3	F	N		23.0	22.0	23.0
4	F	CORR	A	1.0	0.8	0.3
5	F	CORR	B	0.8	1.0	0.6
6	F	CORR	C	0.3	0.6	1.0
7	M	MEAN		12.3	33.4	7.6
8	M	STD		1.1	2.9	0.9
9	M	N		31.0	33.0	32.0
10	M	CORR	A	1.0	0.4	0.8
11	M	CORR	B	0.4	1.0	0.3
12	M	CORR	C	0.8	0.3	1.0

Creating a TYPE=CORR data set in a DATA step: example 3 This example creates a TYPE=CORR data set by reading an input correlation matrix in a DATA step. The matrix is for three variables. **Output 17.3** shows the new data set.

```
DATA CORRMATR(TYPE=CORR);
   INPUT (A B C) (10.7);
   _TYPE_='CORR';
   LENGTH _NAME_ $ 8.;
   IF_N_=1 THEN _NAME_='A';
   IF_N_=2 THEN _NAME_='B';
   IF_N_=3 THEN _NAME_='C';
   CARDS;
1.0000000 0.6198688 0.5297345
0.6198688 1.0000000 0.4292545
0.5297345 0.4292545 1.0000000
;
PROC PRINT;
TITLE 'TYPE=CORR Data Set';
```

Output 17.3 A TYPE=CORR SAS Data Set Created By a DATA Step

TYPE=CORR Data Set 1

OBS	A	B	C	_TYPE_	_NAME_
1	1.00000	0.61987	0.52973	CORR	A
2	0.61987	1.00000	0.42925	CORR	B
3	0.52973	0.42925	1.00000	CORR	C

TYPE=COV Data Sets

A TYPE=COV data set is similar to a TYPE=CORR data set, except that it has _TYPE_='COV' observations rather than _TYPE_='CORR' observations, and it contains a covariance matrix rather than a correlation matrix. COV data sets are created by PROC PRINCOMP if the COV option is specified. PROC CORR produces COV data sets if the COV and NOCORR options are specified and the OUT= data set is assigned TYPE=COV with the TYPE= data set option. For example:

```
PROC CORR COV NOCORR OUT=CVMTRX(TYPE=COV);
```

TYPE=COV data sets are used by these procedures: CANCORR, CANDISC, FACTOR, PRINCOMP, and VARCLUS.

TYPE=SSCP Data Sets

TYPE=SSCP data sets are used to store the uncorrected sums of squares and crossproducts for variables. TYPE=SSCP data sets are produced automatically by PROC REG when OUTSSCP= is specified in the PROC REG statement. You can also create TYPE=SSCP data sets in a DATA step, and in this case TYPE=SSCP must be specified as a data set option.

Variables in a TYPE=SSCP data set If a BY statement is used with PROC REG, the BY variable(s) is first in the TYPE=SSCP data set. The next variable is a special character variable, _NAME_, eight characters long, whose values are the variable names in the original data set from which the SSCP matrix was computed. The next variable is INTERCEP, whose values are the variable sums. Finally come the variables from the original data set that appear in the VAR statement or a MODEL statement.

Observations in a TYPE=SSCP data set For the first observation in the TYPE=SSCP data set, the _NAME_ variable's value is 'INTERCEP'. The value of the INTERCEP variable for this first observation is the number of observations in the original data set. The values of the remaining variables for the INTERCEP observation are the sums of the variables. For the second and following observations in the TYPE=SSCP data set, the _NAME_ variable's value is the name of the corresponding variable in the original data set. The INTERCEP variable's values contain the variable sums. The other variables' values are the sums of the products for the variables.

Using REG with a BY statement: example 4 **Output 17.4** shows a TYPE=SSCP data set created when PROC REG was used with a BY statement. The variables were X and Y.

Output 17.4 A TYPE=SSCP SAS Data Set with BY Variable

OBS	STATE	_NAME_	INTERCEP	X	Y
1	NC	INTERCEP	7.0	350.6	126.6
2	NC	X	350.6	17722.8	6416.9
3	NC	Y	126.6	6416.9	2334.5
4	VA	INTERCEP	8.0	413.1	114.3
5	VA	X	413.1	21484.9	6058.8
6	VA	Y	114.3	6058.8	1811.1

As you can see in the output, for STATE='NC', 7 is the number of observations, 350.6 and 126.6 are the sums of X and Y respectively; 17722.8 and 2334.5 are the sums of X2 and Y2; and 6416.9 is the sum of the crossproducts of X and Y. If a WEIGHT statement is used with PROC REG (for weighted least squares analyses), then the sum of the weights replaces the number of observations in the TYPE=SSCP data set, and all sums of products are weighted.

TYPE=FACTOR Data Sets

TYPE=FACTOR data sets, created automatically by PROC FACTOR when an output data set is specified, contain information about factor analyses. PROC FACTOR and PROC SCORE use TYPE=FACTOR data sets as input.

Variables in a TYPE=FACTOR data set The variables in a TYPE=FACTOR data set correspond to those in a TYPE=CORR data set: BY variables, if any; _TYPE_; _NAME_; and the names of the variables used by PROC FACTOR.

Observations in a TYPE=FACTOR data set Each observation in the output data set contains some type of statistic as indicated by the _TYPE_ variable. The _NAME_ variable is blank except where otherwise indicated. The values of the _TYPE_ variable are as follows:

TYPE	Contents
MEAN	means.
STD	standard deviations.
N	sample size.
CORR	correlations. The _NAME_ variable contains the name of the variable corresponding to each row of the correlation matrix.
IMAGE	image coefficients. The _NAME_ variable contains the name of the variable corresponding to each row of the image coefficient matrix.
IMAGECOV	image covariance matrix. The _NAME_ variable contains the name of the variable corresponding to each row of the image covariance matrix.
COMMUNAL	final communality estimates.
PRIOR	prior communality estimates, or estimates from the last iteration for iterative methods.
WEIGHT	variable weights.
EIGENVAL	eigenvalues.
UNROTATE	unrotated factor pattern. The _NAME_ variable contains the name of the factor.

RESIDUAL	residual correlations. The _NAME_ variable contains the name of the variable corresponding to each row of the residual correlation matrix.
TRANSFOR	transformation matrix from rotation. The _NAME_ variable contains the name of the factor.
FCORR	inter-factor correlations. The _NAME_ variable contains the name of the factor.
PATTERN	factor pattern. The _NAME_ variable contains the name of the factor.
RCORR	reference axis correlations. The _NAME_ variable contains the name of the factor.
REFERENC	reference structure. The _NAME_ variable contains the name of the factor.
STRUCTUR	factor structure. The _NAME_ variable contains the name of the factor.
SCORE	scoring coefficients. The _NAME_ variable contains the name of the factor.

For an example of a TYPE=FACTOR data set, see the description of PROC FACTOR in *SAS User's Guide: Statistics*.

TYPE=EST Data Sets

TYPE=EST data sets, produced by PROC REG when the OUTEST= option is specified, store the coefficients of linear models.

Variables in a TYPE=EST data set If a BY statement is used with PROC REG, the BY variables appear first in the TYPE=EST data set. Next comes the character variable _TYPE_, eight characters long, which indicates the source of the coefficients (identity, OLS, 2SLS). The variable _MODEL_ is next, and it contains the label that is associated with the MODEL statement in PROC REG. _DEPVAR_ contains the name of the dependent variable. Next is the variable _SIGMA_, which contains the standard deviation for the error term. The regression coefficients of the variables in the model appear next in the order they first appear in a MODEL statement. The last variable in the data set is INTERCEP, which contains the equation constant.

If the COVOUT option is used in PROC REG, the covariance matrix of the estimates is output after the parameter estimates. The value of the _TYPE_ variable for these observations is COVB, and the rows are identified by the value of the _MODEL_ variable.

Observations in a TYPE=EST data set The TYPE=EST data set has one observation for each dependent variable on the left-hand side of a MODEL statement specified with PROC REG.

Creating a TYPE=EST data set: example 5 **Output 17.5** shows the TYPE=EST data set produced by the PROC REG step below.

```
PROC REG OUTEST=B;
   L1: MODEL Y=XW;
   L2: MODEL W=Z;
PROC PRINT;
TITLE 'TYPE=EST Data Set';
```

Output 17.5 A TYPE=EST SAS Data Set

TYPE=EST Data Set 1

OBS	_TYPE_	_MODEL_	_DEPVAR_	_SIGMA_	Y	X	W	Z	INTERCEP
1	OLS	L1	Y	0.567	-1	0.234	0.045	.	4.3338
2	OLS	L2	W	0.400	.	.	-1.000	0.309	2.9877

TYPE=DISCAL Data Sets

The TYPE=DISCAL data set contains calibration information developed by PROC DISCRIM. TYPE=DISCAL data sets can be used by PROC DISCRIM to classify observations in other data sets.

Variables in a TYPE=DISCAL data set The first variable in a TYPE=DISCAL data set is the _TYPE_ variable; its values identify the type of each observation in the data set. The second variable is the variable specified in the CLASS statement of PROC DISCRIM. _LNDET_ is the third variable; it contains the log determinant of the covariance matrix. The next variable is _PRIOR_, the prior probability of classification membership. The remaining variables are those specified in the VAR statement of PROC DISCRIM.

Observations in a TYPE=DISCAL data set The first observation has one of four values for _TYPE_: PLEQ, PLPR, NOEQ, or NOPR. These values indicate whether the analysis is based on the pooled (PL) or within-group (NO) covariance matrix, and whether the prior probabilities are equal (EQ) or proportional (PR). The next observation contains the means for all variables listed in the VAR statement (_TYPE_='MEAN'). If POOL=YES, all _TYPE_='MEAN' observations are printed before the remaining observation types. The next observation contains the standard deviation (_TYPE_='STD'). There is one _TYPE_='STD' observation if POOL=YES; if POOL=NO there are as many STD observations as levels of the CLASS variable. The remaining observations are _TYPE_='RINV'; one for each row of each correlation inverse matrix. There is one matrix if POOL=YES. If POOL=NO, there is a matrix for each level of the CLASS variable.

A DISCAL data set with POOL=NO: example 6 **Output 17.6** shows a TYPE=DISCAL data set produced by PROC DISCRIM with POOL=NO.

Output 17.6 A TYPE=DISCAL SAS Data Set with POOL=NO

REMOTE SENSING DATA ON FIVE CROPS
CLASSIFICATION OF CROP DATA

OBS	_TYPE_	CROP	_LNDET_	_PRIOR_	X1	X2	X3	X4
1	NOEQ		5.0000	.	.	.	.	.
2	MEAN	CLOVER	23.6462	0.2	46.364	32.636	34.182	36.636
3	STD	CLOVER	23.6462	0.2	25.905	17.078	20.517	20.568
4	RINV	CLOVER	23.6462	0.2	1.294	-0.002	-0.557	-0.450
5	RINV	CLOVER	23.6462	0.2	-0.002	1.346	-0.653	0.091
6	RINV	CLOVER	23.6462	0.2	-0.557	-0.653	1.633	0.435
7	RINV	CLOVER	23.6462	0.2	-0.450	0.091	0.435	1.238
8	MEAN	CORN	11.1347	0.2	15.286	22.714	27.429	33.143
9	STD	CORN	11.1347	0.2	1.799	6.448	5.623	18.632

(continued on next page)

(continued from previous page)

10	RINV	CORN	11.1347	0.2	3.768	1.063	1.815	5.153
11	RINV	CORN	11.1347	0.2	1.063	2.055	0.372	2.486
12	RINV	CORN	11.1347	0.2	1.815	0.372	4.146	5.115
13	RINV	CORN	11.1347	0.2	5.153	2.486	5.115	10.915
14	MEAN	COTTON	13.2357	0.2	34.500	32.667	35.000	39.167
15	STD	COTTON	13.2357	0.2	9.566	8.664	19.890	20.478
16	RINV	COTTON	13.2357	0.2	49.877	1.466	-53.645	-14.7044
17	RINV	COTTON	13.2357	0.2	1.466	36.516	-39.363	-16.9000
18	RINV	COTTON	13.2357	0.2	-53.645	-39.363	97.963	33.239
19	RINV	COTTON	13.2357	0.2	-14.704	-16.900	33.239	12.888
20	MEAN	SOYBEANS	12.4526	0.2	21.000	27.000	23.500	29.667
21	STD	SOYBEANS	12.4526	0.2	5.060	10.714	5.128	11.843
22	RINV	SOYBEANS	12.4526	0.2	10.861	-2.052	-6.386	5.558
23	RINV	SOYBEANS	12.4526	0.2	-2.052	4.287	-1.082	1.377
24	RINV	SOYBEANS	12.4526	0.2	-6.386	-1.082	6.097	-4.675
25	RINV	SOYBEANS	12.4526	0.2	5.558	1.377	-4.675	5.355
26	MEAN	SUGARBEETS	17.7629	0.2	31.000	32.167	20.000	40.500
27	STD	SUGARBEETS	17.7629	0.2	11.967	13.152	10.257	16.489
28	RINV	SUGARBEETS	17.7629	0.2	2.020	-0.454	-2.003	-2.569
29	RINV	SUGARBEETS	17.7629	0.2	-0.454	3.091	4.098	3.572
30	RINV	SUGARBEETS	17.7629	0.2	-2.003	4.098	8.702	7.895
31	RINV	SUGARBEETS	17.7629	0.2	-2.569	3.572	7.895	8.534

A DISCAL data set with POOL=YES: example 7 **Output 17.7** shows a TYPE=DISCAL data set produced by PROC DISCRIM using the same original data set, but with POOL=YES.

Output 17.7 A TYPE=DISCAL SAS Data Set with POOL=YES

REMOTE SENSING DATA ON FIVE
CLASSIFICATION OF CROP DATA

OBS	_TYPE_	CROP	_LNDET_	_PRIOR_	X1	X2	X3	X4
1	PLEQ		5	.	.	.	.	.
2	MEAN	CLOVER	.	0.2	46.3636	32.6364	34.1818	36.6364
3	MEAN	CORN	.	0.2	15.2857	22.7143	27.4286	33.1429
4	MEAN	COTTON	.	0.2	34.5000	32.6667	35.0000	39.1667
5	MEAN	SOYBEANS	.	0.2	21.0000	27.0000	23.5000	29.6667
6	MEAN	SUGARBEETS	.	0.2	31.0000	32.1667	20.0000	40.5000
7	STD		.	.	16.0960	12.6748	15.0642	18.3787
8	RINV		.	.	1.2881	-0.0463	-0.5969	-0.4027
9	RINV		.	.	-0.0463	1.2270	-0.4620	0.1129
10	RINV		.	.	-0.5969	-0.4620	1.6007	0.5365
11	RINV		.	.	-0.4027	0.1129	0.5365	1.2672

NOTES

1. **AOS/VS, PRIMOS, and VMS**: User-written formats are SAS files. See PROC FORMAT for details.

CMS, OS, VM/PC, and VSE: User-written formats are **not** SAS files in these operating environments. They are stored as external files. See the description of PROC FORMAT for complete information on user-written formats and where they can be stored.

2. **AOS/VS, PRIMOS, and VMS**: Note that the extension or suffix to the file's name (as listed by the operating system) indicates what type of SAS file it is. The extensions (suffixes) are:

- SFC (permanent character format)
- SFN (permanent numeric format)
- SFX (temporary character format)
- SFW (temporary numeric format)

- SSD (permanent SAS data set)
- SSW (temporary SAS data set)
- SFS (catalog)
- SSG (graphics catalog)
- SET (model)
- SIM (IML work space)

CMS and VM/PC: SAS maintains information on the type of each file by automatically adding a prefix to the second-level name. You do not see the prefix in SAS output, and you do not need to use the prefix to reference the SAS file in SAS programs. If you use a CMS command such as FILELIST to display a listing of files, you will notice that the prefix appears with the file's name.

OS and VSE: SAS maintains information on the type of each file by automatically adding a prefix to the second-level name. You do not see the prefix in SAS output, and you do not need to use the prefix to reference the SAS file in SAS programs.

3. **AOS/VS, PRIMOS, and VMS**: SAS format files are an exception to this general rule because the libref is assigned in the PROC FORMAT statement, but the second-level name is the format's name and is assigned in a VALUE or PICTURE statement.

4. **AOS/VS, PRIMOS, and VMS**: Second-level names can be eight characters long for any type of SAS file.

CMS, OS, VM/PC, and VSE: The second-level name of several kinds of SAS files is restricted to seven characters under these operating systems. SAS files that can have only seven-character names are model, catalog, graphics catalog, and work space files. SAS data sets can have eight-character names.

5. **AOS/VS**: Use the SAS statement LIBNAME to associate a libref with the name of the directory in which you want to store or have stored a SAS file. For example, suppose you are creating the SAS data set called PRICES, and you decide to store it in a directory called :UDD:SURVEY:STORES. Before the DATA step that creates PRICES, enter a LIBNAME statement:

```
LIBNAME FOOD ':UDD:SURVEY:STORES';
DATA FOOD.PRICES;
   INPUT ... ;
more SAS statements
```

Similarly, if you want to access an existing SAS file, you use the LIBNAME statement to associate a libref with the directory in which the file is stored. For example

```
LIBNAME FOOD ':UDD:SURVEY:STORES';
DATA SUPRMART;
   SET FOOD.PRICES;
   IF STORE= ... ;
more SAS statements
```

A directory does not have to be referred to by the same libref all the time. A different libref could be associated with a directory in every SAS job using the directory.

The libref of a transport-format SAS file (a SAS file that is in a special format for moving to another computer) cannot be defined with the LIBNAME statement. Use the FILENAME statement instead.

CMS and VM/PC: Under most circumstances, you do not need to explicitly associate a libref with a SAS file because SAS makes the association automatically.

When you are creating a new SAS file and you specify a two-level SAS file name, SAS takes the following steps:

1. First, SAS checks to see if a FILEDEF has been issued with a DDname that matches the libref you used. If such a FILEDEF is in effect, SAS writes the new SAS file to the minidisk indicated by the FILEDEF, using the libref as the file's filetype and the second-level name as the file's filename.
2. If no matching FILEDEF is found, SAS searches all write-accessed minidisks to see if there are any files with filetypes that match the libref. If one is found, SAS writes the new SAS file to that minidisk, using the libref as the file's filetype, and the second-level name as the file's filename. (This means that you should not use a libref that matches the filetype of a non-SAS file.) SAS also automatically issues a FILEDEF using the libref as both the DDname and the filetype, and the second-level name as the filename.
3. If there is no file with a matching filetype, SAS issues a FILEDEF using the libref for the DDname parameter and for the filetype. By default, the file will be written to your A-disk.

For example, suppose you are creating a SAS data set called FOOD.PRICES:

```
DATA FOOD.PRICES;
   INPUT ... ;
more SAS statements
```

When SAS reads the name FOOD.PRICES, it checks to see if a FILEDEF with DDname FOOD has been issued. If there is no such FILEDEF, SAS searches all minidisks that are accessed in write mode for filetype FOOD. If no files with filetype FOOD exist, SAS issues a FILEDEF automatically, using FOOD for both the DDname and filetype, PRICES as the filename, and A as the filemode:

```
FILEDEF FOOD DISK PRICES FOOD A
```

If you want a new SAS file to be stored on a disk other than one of the defaults described above, you must issue the appropriate FILEDEF explicitly. Specify the appropriate values for the DDname and filemode parameters. You can specify any value for filename and filetype; SAS will substitute the libref and second-level name for filetype and filename, respectively. For example, if you want to store FOOD.PRICES on a B-disk, your FILEDEF could be

```
FILEDEF FOOD DISK DUMMY DUMMY B
```

Because a SAS file's libref is used for the CMS filetype, you must use the same libref each time you access a SAS file.

To access an existing SAS file on any minidisk, just use the file's name in the appropriate SAS statement. SAS searches all accessed minidisks for the file you have specified and issues the appropriate FILEDEF when it finds the file. For example, suppose FOOD.PRICES is stored on your B-disk. When SAS reads this statement:

```
SET FOOD.PRICES;
```

it searches all accessed minidisks for a file with filename PRICES and filetype FOOD. When SAS finds the file PRICES FOOD B, this FILEDEF is automatically issued

```
FILEDEF FOOD DISK PRICES FOOD B
```

Note: if you have multiple SAS files with the same filename and filetype on different minidisks, SAS will find only the file on the minidisk that is first in the search order. For example, if you have a PRICES FOOD A and a PRICES FOOD B, SAS will find PRICES FOOD A. You must issue a FILEDEF for PRICES FOOD B explicitly if you want to access it.

OS: Associate a libref with your SAS file by using a DD statement (for OS batch) or an ALLOCATE command (for TSO). Specify the libref for the DDname parameter. The OS data set named in the DD statement or ALLOCATE command must be a SAS data library; that is, the OS data set must contain **only** SAS files. (See the *SAS Companion for OS Operating Systems and TSO* for information on creating a SAS data library.)

Suppose you want to create a SAS data set called PRICES, and you want to store it in an existing SAS data library called IDMES.SAS.FILES. You decide to use FOOD as the libref. Under OS batch, you would use a DD statement. For example

```
//FOOD DD DSN=IDMES.SAS.FILES,DISP=OLD
```

Under TSO, use an ALLOC command. For example

```
ALLOC F(FOOD) DA('IDMES.SAS.FILES') OLD
```

To access an existing SAS file, issue a DD statement or ALLOC command for the appropriate SAS data library using the libref as the DDname. For example, to execute a DATA step with a SET statement like this

```
SET FOOD.PRICES;
```

you must first use the appropriate control language:

```
//FOOD DD DSN=IDMES.SAS.FILES,DISP=...
```

Under TSO, use an ALLOC command such as

```
ALLOC F(FOOD) DA('IDMES.SAS.FILES') ...
```

A SAS data library does not have to be referred to by the same libref all the time. A different libref could be associated with a library in every SAS job using the library.

PRIMOS: Use the SAS statement LIBNAME to associate a libref with the name of the directory in which you want to store or have stored a SAS file. For example, suppose you are creating the SAS data set called PRICES, and you decide to store it in a directory called <MASTER1>SURVEY. Before the DATA step that creates PRICES, enter a LIBNAME statement:

```
LIBNAME FOOD '<MASTER1>SURVEY';
DATA FOOD.PRICES;
   INPUT ... ;
more SAS statements
```

Similarly, if you want to access an existing SAS file, you use the LIBNAME statement to associate a libref with the directory in which the file is stored. For example

```
LIBNAME FOOD '<MASTER1>SURVEY';
DATA SUPRMART;
   SET FOOD.PRICES;
   IF STORE= ... ;
more SAS statements
```

A directory does not have to be referred to by the same libref all the time. A different libref could be associated with a directory in every SAS job using the directory.

The libref of a transport-format SAS file (a SAS file that is in a special format for moving to another computer) cannot be defined with the LIBNAME statement. Use the FILENAME statement instead.

VM/PC: See the discussion under CMS.

VMS: Use the SAS statement LIBNAME to associate a libref with the name of the directory in which you want to store or have stored a SAS file. For example, suppose you are creating the SAS data set called PRICES, and you decide to store it in a directory called [MASTER1.SURVEY]. Before the DATA step that creates PRICES, enter a LIBNAME statement:

```
LIBNAME FOOD '[MASTER1.SURVEY]';
DATA FOOD.PRICES;
   INPUT ... ;
more SAS statements
```

Similarly, if you want to access an existing SAS file, you use the LIBNAME statement to associate a libref with the directory in which the file is stored. For example

```
LIBNAME FOOD '[MASTER1.SURVEY]';
DATA SUPRMART;
  SET FOOD.PRICES;
  IF STORE= ... ;
more SAS statements
```

A directory does not have to be referred to by the same libref all the time. A different libref could be associated with a directory in every SAS job using the directory.

The libref of a transport-format SAS file (a SAS file that is in a special format for moving to another computer) cannot be defined with the LIBNAME statement. Use the FILENAME statement instead.

VSE: To associate a libref with a SAS file, you define the SAS data library for the file in your control language, as usual, and use the libref for the filename parameter. For example, if you want to create a SAS data set called PRICES, with a libref of UFOOD, and store it in a SAS data library called IDMES.SAS.FILES, use JCL such as this for a VSE batch job:

```
control language
// DLBL UFOOD,'IDMES.SAS.FILES', ...
// EXTENT ...
// ASSGN ...
more control language
```

Under ICCF, add a FILE command to the proc you use to invoke SAS. For example

```
/FILE NAME=UFOOD,ID'IDMES.SAS.FILE', ...
```

See the *SAS Companion for the VSE Operating System* for important guidelines on naming conventions for VSE SAS files and creating SAS data libraries under VSE.

6. **AOS/VS, OS, PRIMOS, VMS, and VSE**: The SAS System in these environments does not reserve any additional librefs.

CMS and VM/PC: Additional reserved librefs for CMS SAS programs are: MAPS, SORTWKxx, SORTLIB, SYSIN, $SYSLIB, SYSOUT.

7. **AOS/VS, PRIMOS, and VMS**: The LIBSEARCH statement is another exception to the rule of specifying both names to access a permanent SAS file. Note, however, that it only affects reading existing files, not writing new files. See the LIBSEARCH statement for details. Note that if both the USER= option and a LIBSEARCH statement are used, LIBSEARCH specifications supersede the USER= value.

8. See note 5.

9. See note 7.

10. **AOS/VS, PRIMOS, and VMS**: In these environments the "SAS data library" is a logical concept. SAS files can be stored in any directory, along with non-SAS files. All SAS files in a given directory are members of one library. Note that although it is possible to assign more than one libref to the same directory (and therefore, to the same library), in this discussion assume that different librefs always refer to different directories.

CMS and VM/PC: In these environments the "SAS data library" is a logical concept. SAS files can be stored in any minidisk, along with non-SAS files. All SAS files on a given minidisk having the same first-level name (libref) are members of one library. Since the libref of a SAS file becomes the file's filetype, you can group SAS files on a minidisk by assigning the same libref/filetype to them. Members of a library can be processed as a group by the COPY, CONTENTS, and DATASETS procedures.

OS and VSE: In these environments, the SAS data library is a physical entity. A SAS data library is an OS data set or VSE system file that is defined with special internal characteristics and that contains only SAS files. The library includes a directory created automatically by SAS that is used to store information about the members of the library. You create (allocate space for) a new SAS data library with job control language, either prior to or accompanying the SAS job or session that creates the first SAS data set in the library. See the SAS Companion for your operating system for instructions on creating SAS data libraries.

11. See note 10.

12. **AOS/VS, PRIMOS, and VMS**: In these environments, only SAS data sets can be stored on tape. Catalogs, graphics catalogs, formats, model, and saved work space files cannot be stored on tape. Note that the LIBNAME statement for a tape SAS data set must specify a tape device name rather than a directory name.

VM/PC: Since the XT/370 does not support tape drives, SAS data libraries cannot be stored on tape with VM/PC SAS.

CMS, OS, and VSE: Any type of SAS file can be stored on tape.

13. **CMS, OS, and VSE**: SAS data sets on tape created by any one of these operating systems can be processed by SAS programs on either of the other two operating systems. For example, a CMS SAS data set can be read by OS SAS and VSE SAS programs.

14. See note 12.

15. **OS and VSE**: When you add a new SAS file to a SAS data library on tape, it is written at the end of the data library. In addition, when you replace a SAS data set on tape with another SAS data set that has the same name, the original data set is over-written and any other files after it on the tape are erased. This is true even if PROTECT= passwords have been assigned to the other SAS data sets on the tape. (This is a limitation of tapes rather than SAS.)

Never use the DCB parameters RECFM, LRECL, BLKSIZE, BFTEK, or BUFNO for tape SAS data sets. If you want to override the default block size or buffer number, use SAS data set options, for example,

```
DATA M.SURVEY(BLKSIZE=32000);
```

16. **AOS/VS, PRIMOS, and VMS**: Tape-format or sequential SAS data sets on disk are supported in these environments. Other types of SAS files cannot be written in tape-format on disk. If you store a SAS data set on disk in tape-format, it can be read by the SAS System on another computer with the same operating system, if a network exists. For example, a VMS SAS data set on one VAX can be read by the VMS SAS System on another VAX, if there is a network for disk sharing.

CMS, OS, and VM/PC: Tape-format SAS files on disk are supported. See the Companion for your operating system for more information on using this file format. The sequential nature of the tape format has limitations, but sometimes

tape-format disk SAS data libraries on one of these operating systems can be processed by SAS software on the other two operating systems, if the computer installation supports a shared-disk network. For example, a tape-format OS SAS data library can be read by CMS SAS. This feature is useful when direct-access data sets are not handled by local utility programs, or when SAS libraries need to be interchanged between processors and/or operating systems.

VSE: Tape-format SAS files on disk are not supported.

17. **OS and VSE**: If you are creating a SAS data set that has the same name as an existing SAS data set in your SAS data library, the library must have enough free space to hold a copy of the SAS data set being replaced. The original data set is deleted only **after** the new data set has been written completely with no errors.

VMS: When REPLACE is in effect and you have allowed for more than one version of a file to be maintained, the original SAS file is not erased, but becomes version 1. The new SAS file is version 2, the current version.

18. **AOS/VS, PRIMOS, and VMS**: The REPLACE= data set option is not supported, however, the REPLACE/NOREPLACE system option is supported.

19. **AOS/VS, CMS, PRIMOS, VM/PC, and VMS**: You can use system commands or utilities to copy SAS files as well.

OS and VSE: Do not copy SAS data libraries with anything but PROC COPY. If you use a non-SAS utility to copy SAS files, the files will be unreadable.

20. See note 12.

21. **AOS/VS and VMS**: The libref of a transport-format file cannot be defined with the SAS LIBNAME statement. Instead, AOS/VS users should define the libref with the CREATE/LINK commmand, and VMS users should define the libref with the ASSIGN command.

22. **AOS/VS and PRIMOS**: You must use IXTAPE as the libref of a transport-format SAS library. Use the FILENAME statement rather than the LIBNAME statement to define the libref.

CMS, OS, and VSE: You can use any nonreserved libref for transport libraries.

VMS: You must use IXTAPE as the libref of a transport-format SAS library. Use the FILENAME statement rather than the LIBNAME statement to define the libref. Also note that when the transport library is on tape, you must use the VMS ASSIGN command to associate the libref IXTAPE with the tape drive, and the logical name of the tape drive must be SASTAPE:

```
ASSIGN SASTAPE IXTAPE
```

23. **AOS/VS, CMS, PRIMOS, VM/PC, and VMS**: You can use operating system utilities to transport SAS files between computers, as long as you are transporting to a computer running the same operating system. For example, you can move files from one CMS machine to another.

OS and VSE: Never attempt to process SAS files with operating system utilities. The files will be destroyed.

24. **AOS/VS, PRIMOS, and VMS**: No history information is maintained for SAS files under these operating systems.

25. See note 24.

26. **AOS/VS, PRIMOS, and VMS**: A SAS data set can contain up to 2^{31} observations, and the observations can be any length.

CMS and VM/PC: A SAS data set can contain any number of observations but cannot exceed one tape or disk volume in size. An observation cannot contain more than approximately 32,000 bytes of information (for example, no more than about 4000 variables with lengths of eight bytes). The exact upper limit varies, depending on the number of automatic variables that SAS creates in the step.

OS and VSE: A SAS data set can contain any number of observations, but disk SAS data sets cannot exceed more than one disk volume in size. An observation cannot contain more than approximately 32,000 bytes of information (for example, no more than about 4000 variables with lengths of eight bytes). The exact upper limit varies and depends upon the track size of the device on which the data set is stored and the number of automatic variables that SAS creates in the step.

27. **OS**: The FILECLOSE=FREE specification has an effect only under MVS.

VSE: The only valid values are REREAD, REWIND, and LEAVE. DISP and FREE cannot be used.

28. See note 27.

29. See note 27.

30. **CMS**: When FILEDISP=NEW, SAS assumes that the tape to which you are writing is new and begins writing the new SAS data set where the tape is positioned. When FILEDISP=OLD, SAS assumes that the tape has a SAS data library on it and reads until the end-of-file is reached before writing the new SAS data set.

OS: FILEDISP=NEW overrides a disposition specification of OLD in the control language for your job, but FILEDISP=OLD does not override a specification of NEW in the control language.

When FILEDISP=NEW, SAS assumes that the data library is empty and does not look for previously-written SAS data sets. For as long as the data library is allocated (which could span several SAS sessions) SAS knows that the data library is not empty, even if you subsequently specify FILEDISP=NEW. When FILEDISP=OLD, SAS looks for other SAS files and writes the new data set at the end of the data library.

FILEDISP=OLD specifies that the tape-format data library is not initially empty.

VSE: When FILEDISP=NEW, SAS assumes that the data library is empty and does not look for previously-written SAS data sets. For as long as the data library is allocated (which could span several SAS sessions) SAS knows that the data library is not empty, even if you subsequently specify FILEDISP=NEW. When FILEDISP=OLD, SAS looks for other SAS files and writes the new data set at the end of the data library.

Chapter 18

SAS® System Options

INTRODUCTION

This chapter describes the *SAS system options*, which are specified in the OPTIONS statement or when SAS is invoked.

SAS software uses two other kinds of options in addition to the SAS system options:

- *SAS data set options* (such as RENAME= and KEEP=), which are specified in parentheses following a SAS data set's name and affect only that SAS data set
- *statement options* (such as HEADER= for the FILE statement), which are specified only in a given SAS statement or statements and affect only that statement or step.

The SAS system options differ from SAS data set or statement options because, once invoked, they are in effect for all DATA and PROC steps in a SAS job or session unless they are respecified in another OPTIONS statement. For example, the CENTER | NOCENTER option affects all of the output from a SAS job, regardless of how many DATA and PROC steps it contains.

The current settings for the SAS system options are reported by the OPTIONS procedure. (SAS data set and statement options are **not** reported by PROC OPTIONS.) When SAS software is sent to a computer installation, defaults are set for each system option. Your computer installation can alter these defaults if necessary. The defaults described for the SAS system options in this chapter reflect the option settings as shipped by SAS Institute. You should execute PROC OPTIONS to check the default settings at your installation.

If you need to alter your installation's default for an option or options, specify the option in an OPTIONS statement or when invoking SAS. Note that your installation has the ability to *restrict* many of the SAS system options, if it feels it is necessary for security or efficiency. If you attempt to change a restricted option, you receive a message indicating that the option's default cannot be overridden.

Many of the SAS system options have different defaults under different operating systems. In addition, some options' defaults are different for batch execution and other modes of execution. So that you can easily find the default information for options in your operating environment, tables have been provided for each

operating system. The tables list each option, its default value (as shipped from SAS Institute), and whether it can be specified when SAS is invoked or in an OPTIONS statement.[1] These tables can be found following the option descriptions in the next section.

OPTION DESCRIPTIONS

This section lists the SAS system options in alphabetical order. Not all options apply to all operating systems, so refer to the operating system information at the end of each option description to be sure the option applies to your SAS jobs. You should also refer to the table for your operating system to check default settings and to see whether the option can be specified in the OPTIONS statement, when SAS is invoked, or in both places. The operating system tables follow this list of option descriptions.

This list does not include options that are specific to SAS/GRAPH software. Refer to the *SAS/GRAPH User's Guide* for descriptions of all SAS/GRAPH options.

The SAS System includes two sets of SAS system option default values. One is the BATCH set, typically used for SAS jobs run in batch execution mode. The other is the INTERACTIVE set, used for SAS jobs run in interactive mode. (See **Table 18.1** for details for your operating system.) Separate option default tables are provided because many installations prefer to use different defaults for different execution modes. Refer to the "Operating System Notes" Appendix or to the *SAS Companion* for your operating system for more information about modes of execution.

Table 18.1 Default Use of BATCH and INTERACTIVE Options

Operating System	SAS Execution Mode			
			Interactive	
	Batch	Noninteractive	Line Mode	Display Manager
AOS/VS	BATCH	BATCH	INTERACTIVE	INTERACTIVE
CMS	BATCH	BATCH	INTERACTIVE	INTERACTIVE
PRIMOS	BATCH	BATCH	INTERACTIVE	INTERACTIVE
OS	BATCH	INTERACTIVE	INTERACTIVE	INTERACTIVE
VM/PC	N/A	BATCH	INTERACTIVE	INTERACTIVE
VMS	BATCH	BATCH	INTERACTIVE	INTERACTIVE
VSE	BATCH	INTERACTIVE	INTERACTIVE	INTERACTIVE

BATCH/INTERACTIVE
specifies which set of SAS system option default values is in effect when the SAS System executes.

Note that the setting for this option does not specify the mode of execution, only the set of option defaults to be used. By default, SAS

uses the two sets of options as shown in **Table 18.1**. However, you can specify BATCH to use the BATCH table in an interactive SAS job. You cannot use the INTERACTIVE table in a batch job.

Operating systems: All

BAUD=*rate*
specifies the communications line data transmission rate. BAUD should be set to the baud rate (typically 300 or 1200) that most asynchronous ASCII display or graphics terminals at your installation use. SAS/GRAPH software, SAS/FSP software, and the CLEAR statement use this option.

Operating systems: CMS, OS, VM/PC, and VSE

BLDLTABLE/NOBLDLTABLE
BLDL/NOBLDL
specifies whether the dynamic BLDL table facility is to be active. The program management component of the SAS supervisor optionally maintains a BLDL table of loaded modules that provides faster execution and reduces I/O activity on STEPLIB and TASKLIB, including SAS.LIBRARY. If SAS.LIBRARY or any STEPLIB data set is to be modified (for example, compressed) during the execution of a SAS job, NOBLDLTABLE should be specified.

Operating system: OS

BLKSIZE=*n*
specifies the default block size for SAS data sets.[2] A value of zero indicates that the value given by the BLKSIZE(*devicetype*) option, below, is to be used. Except under CMS and VM/PC, if the storage device in use has a smaller maximum block size than the default specified here, the maximum value for the storage device overrides. The actual block size used for a data set may exceed the BLKSIZE= option value if the length of an observation is greater than the specified value. Depending on the computer system, large block sizes can sometimes increase the efficiency of SAS jobs. The BLKSIZE= value can be zero or range from 1024 to 32760.

Operating systems: CMS, OS, VM/PC, and VSE

BLKSIZE(*devicetype*)=*value*
specifies the default block size for SAS data sets by *devicetype*. Any valid specific device number may be specified for *devicetype*, as well as generic types such as DISK or DASD, TAPE, CMS, FBA, and OTHER.

- DISK, DASD, and TAPE set all of the corresponding values.
- CMS specifies the block size to use for SAS data sets stored on CMS minidisks.
- FBA specifies the logical block size to use for SAS data sets on FBA devices in the VSE environment.
- The SAS System uses OTHER when it is not able to determine the exact device type.

The following can be specified for *value*:

number
specifies the block size that the SAS System is to use for the device.

OPT
: specifies that the SAS System is to choose an optimum block size for the device.

MAX or FULL
: specifies that the SAS System is to use the maximum permitted block size for the device.

HALF, THIRD, FOURTH, or FIFTH
: specifies that the SAS System is to use the largest value that results in obtaining two, three, four, and five, respectively, blocks per track (if a disk device) or the maximum permitted block size divided by two, three, four, and five, respectively (if not a disk device).

The following example tells the SAS System to choose optimum block size values for all disk devices except 3380s, for which one-third track blocking is requested, and to use the maximum permitted block size on all tape devices:

```
OPTIONS BLKSIZE(DISK)=OPT
   BLKSIZE(3380)=THIRD
   BLKSIZE(TAPE)=MAX;
```

The DISK or DASD specification for *devicetype* sets values for the following device types: 2301, 2302, 2303, 2305-1, 2305-2, 2311, 2314, 2321, 3330, 3330-1, 3340, 3350, 3375 and 3380. The TAPE specification for *devicetype* sets values for the 2400 and 3400 device types.

Operating systems: CMS, OS, VM/PC, and VSE

BUFNO=*n*
: gives the number of buffers to use for each SAS data set. Valid values are 1 and 2.

Operating systems: OS and VSE

BYERR/NOBYERR
: controls whether the SAS System generates an error message and sets the error flag when using a _NULL_ data set (or a data set that is bypassed by the operating system control language) in the SORT procedure. Under BYERR, the following message is generated when a null data set is input for the SORT procedure: "No BY statement used or no variables found. A BY statement must be used with PROC SORT to indicate by which variables the data set is to be sorted." This option is used only in PROC SORT, but it must be specified as a system option.

Operating systems: CMS, OS, VM/PC, and VSE

C60/C48/C96
: controls whether SAS output is printed using the 60-character set, the 48-character set, or the 96-character set. The C60 character set is the same as the PL/I character set. The C96 character set is the same as the C60 character set with the addition of the lowercase letters. If all printers on which SAS output is printed and all terminals on which SAS output is displayed have lowercase alphabetic graphics installed or if lowercase alphabetic characters are always translated by system software or by hardware to their upper case equivalents, then specify

C96. Otherwise, specify C60. (At present, the only use of the C96 option within the SAS System is with the LIST and HELP statements.)

Operating systems: All

CAPS/NOCAPS
: determines whether lowercase SAS input (including CARDS and PARMCARDS) is translated to uppercase. If the CAPS option is specified, **all** input to the SAS System is translated to uppercase.

When NOCAPS is in effect, all input **except** the following is translated to uppercase:

- data following CARDS and PARMCARDS statements
- text enclosed in single quotes (and double quotes if the DQUOTE option is in effect)
- titles, footnotes, and values in VALUE statements in PROC FORMAT
- variable labels and data set labels.

Operating systems: All

CARDS=MAX or *n*
C=MAX or *n*
: gives the maximum number of cards a SAS job can punch to the standard SAS punch file (FILE PUNCH). If CARDS=MAX or C=MAX is specified, the SAS System does not limit the number of cards punched. When this option is specified in the OPTIONS statement, the value given cannot exceed the default value or the value specified at SAS invocation. In all cases, local installation limits apply to SAS jobs.

Operating systems: CMS, OS, VM/PC, and VSE

CASORT/NOCASORT
: specifies that the CA-SORT/DOS™* program product is used as the system sort utility.

Operating system: VSE

CENTER/NOCENTER
: controls whether SAS procedure output is centered.

Operating systems: All

CHARCODE/NOCHARCODE
: permits users who do not have an underscore character (_) , vertical bar (|) , logical not sign (¬) or braces ([and]) on their terminals to substitute character combinations for these symbols.[3] This option allows the following substitution of characters:

?- for the underscore
?/ for the vertical bar
?= for the logical not sign
?(for the left brace
?) for the right brace.

Operating systems: All

CHKPT/NOCHKPT
: checkpoints the status and certain SAS options at the end of every SAS DATA or PROC step.[4]

---

* CA-SORT/DOS is a trademark of Computer Associates International, LTD.

Operating systems: CMS, OS, VM/PC, and VSE

CLIST/NOCLIST
: controls whether SAS obtains its input from the terminal directly via TGET (NOCLIST) or via the PUTGET service routine (CLIST) when the SAS System is running interactively under TSO. When CLIST is specified, you can use TSO CLISTs that include SAS statements after the TSO command that invokes the SAS System.

  Operating system: OS

CMDMSG/NOCMDMSG
: specifies whether CMS error messages issued during a SAS session are displayed on the terminal. CMDMSG is helpful when debugging.

  Operating systems: CMS and VM/PC

CPSP/NOCPSP
: determines whether or not the CMS SAS interface issues CP SPOOL commands for the virtual printer. NOCPSP is intended for running the SAS System in a CMS batch environment where spooling commands issued by the SAS System may interfere with those issued by CMS batch facilities.

  If the CPSP and PPrint options are specified, the CMS SAS interface issues

  ```
  CP SPOOL PRINTER CONT
  ```

  when the SAS System is invoked, and

  ```
  CP SPOOL PRINTER NOCONT CLOSE
  ```

  before returning to CMS at the end of the SAS session.

  Operating systems: CMS and VM/PC

DAREAD/NODAREAD
: specifies that the INFILE statement can be used for direct, as well as sequential, read access to disk files.

  Operating system: VSE

DATE/NODATE
: controls whether the current date is printed at the top of each page of the SAS log, the standard SAS print file, and any FILE with the PRINT attribute. If NODATE is specified, the date is omitted from the first title line of each page.

  Operating systems: All

DAUPD/NODAUPD
: specifies that the FILE statement can be used for direct update access to a disk file.

  Operating system: VSE

DEFAULT=*name*
D=*name*
: gives the name of a set of default system options set by your SAS installation representative. D=LOCAL is the standard DEFAULT= value when executing the SAS System under batch, and D=*xxx* (where *xxx* is the name of the interactive monitor) is the standard value when executing the SAS System interactively. The standard value for noninteractive jobs is D=BATCH.

Operating systems: CMS, OS, VM/PC, and VSE

DEVICE=*name*
specifies a terminal device name designation. SAS/GRAPH software, SAS/FSP software, and the CLEAR statement use this option.

Operating systems: All

DISK=*device*
gives the generic or site-dependent name defined at OS system generation for direct access devices.

Operating system: OS

DMS/NODMS
specifies whether the SAS Display Manager System is to be active in a SAS session. This option is totally independent of the FS and FSP options, but you must be using a full-screen terminal to invoke DMS.[5]

Operating systems: All

DQUOTE/NODQUOTE
specifies whether character literals may be bounded by double quotes ("). If the DQUOTE option is specified, character literals of the form "ABCDE" are accepted and may contain unpaired single quotes ('). For example, "DON'T" is an accepted character literal if DQUOTE is specified. However, double quotes that are part of the character value must be paired. This may affect existing TITLE, FOOTNOTE, NOTE, LABEL, and VALUE statements and data set label specifications. If the NODQUOTE option is in effect, character literals must be surrounded by single quotes, and single quotes that are part of the character value must be paired (as in 'DON''T').

Note: to use macro variable references and macro calls within a string enclosed in quotes, you must specify DQUOTE and enclose the string in double quotes. Macro variable references and macro calls within a string enclosed in single quotes (') **are not** expanded.

Operating systems: All

DSNFERR/NODSNFERR
controls whether the SAS System sets the error flag and generates an error message when a SAS data set specified in a job is not found. If DSNFERR is specified and you attempt to reference a SAS data set that does not exist, the SAS System issues an error message and stops processing. If NODSNFERR is specified, this error is ignored and the data set reference is treated as if you had specified _NULL_.

Operating systems: CMS, OS, VM/PC, and VSE

DSRESV/NODSRESV
controls whether certain SAS utility procedures (PDS, PDSCOPY, and RELEASE) issue the RESERVE macro instruction when accessing OS partitioned data sets on shared disk volumes. If DSRESV is specified, the device is reserved, which prevents other processors from accessing the volume on which the PDS resides. If NODSRESV is specified, then the OS-defined resources are enqueued instead.

Operating system: OS

DUMP/NODUMP
controls whether a SAS memory dump is produced if the SAS System terminates abnormally.

The DUMP option invokes a special error handler for SAS procedures so that error diagnostics, system control blocks, and all low subpools of memory are dumped to the SAS log. If there is a SASDUMP DD statement, that file is used instead of the SAS log. Error diagnostics include the abend code, PSWs, program registers at abend, and a trace of the save area that includes the module name and assembler-type offset. Use the DUMP option to report a problem to SAS Institute when you rerun a SAS job that has abended.[6]

Operating system: OS

DUMPM/NODUMPM
DUMPSHORT/NODUMPSHORT
controls whether a short dump is produced when a SAS program abends. This dump is a standard SAS dump without storage subpools and is for diagnostic purposes.

Operating system: OS

DUMPP/NODUMPP
DUMPLONG/NODUMPLONG
controls whether a long dump, which includes program storage, is produced when a SAS program abends. This option is for diagnostic use.

Operating system: OS

DUMPSYS/NODUMPSYS
controls whether the SAS System attempts to disable any interception of program checks and abnormal termination while the SAS System is executing. Specify DUMPSYS if neither the SAS System nor any procedure is to be allowed to intercept program checks and abnormal terminations (abends). This option is for diagnostic use where the DUMP option is ineffective or unusable.[7]

Operating systems: CMS, OS, VM/PC, and VSE

DYNALLOC/NODYNALLOC
controls whether the SAS System (NODYNALLOC) or the system sort utility (DYNALLOC) is to allocate sort work areas dynamically. If you specify DYNALLOC, PROC SORT passes the DYNALLOC=(SORTDEV,SORTWKNO) parameter to the system sort utility.

Operating system: OS

ERASE/NOERASE
specifies whether or not the SAS WORK data sets are erased at the beginning and end of the SAS session. If ERASE is in effect, the CMS SAS interface issues a CMS ERASE for all files with filetype WORK on the disk to be used for the current SAS WORK data library.

Although NOERASE prevents the WORK data sets from being deleted, it has no effect on initialization of the WORK library by the SAS System. The SAS System normally initializes the WORK library at the start of each session—effectively wiping out any pre-existing information. The WORK library initialization can be suppressed by

specifying the SAS system option NOWORKINIT in the SAS command. Consult "SAS Files" for a complete discussion.

Operating systems: CMS and VM/PC

ERRORABEND/NOERRORABEND
When ERRORABEND is in effect, the SAS System abends for most errors (including syntax errors) that normally cause the SAS System to issue an error message, set OBS=0, and go into syntax check mode. The ERRORABEND option is recommended for debugged, production SAS programs that presumably should not encounter any errors at all so that the abend immediately brings to your attention any errors that do occur.

Operating systems: All

ERRORS=*n*
specifies the maximum number of observations for which complete error messages are printed. If data errors are detected in more observations than the number specified, processing continues, but error messages do not print for the additional errors.

Operating systems: All

FILCLR/NOFILCLR
specifies whether the SAS System issues the command FILEDEF * CLEAR during SAS termination. NOFILCLR specifies that only FILEDEFs issued by the SAS System are cleared. FILCLR clears all FILEDEFs except those issued with the PERM option.

Operating systems: CMS and VM/PC

FILEBLKSIZE(*devicetype*)=*value*
specifies, by *devicetype*, the default maximum block size for external files used by certain SAS procedures (for example, PROC SOURCE) and DATA step external files.

Any valid specific device number may be specified for *devicetype*, as well as generic types such as DISK or DASD, TAPE, CMS, FBA, SYSOUT, and OTHER.

- DISK, DASD, and TAPE set all of the corresponding values.
- CMS specifies the block size for external files stored on CMS minidisks. The default value is 32760.
- FBA specifies the logical block size for SAS data sets on FBA devices in the VSE environment.
- SYSOUT specifies the block size for external files whose operating system control language includes a "print" specification, or any external file with the PRINT attribute. The default value is 141.
- OTHER is used when the SAS System is unable to determine the exact device type.

The DISK or DASD specification for *devicetype* sets values for the following device types: 2301, 2302, 2303, 2305-1, 2305-2, 2311, 2314, 2321, 3330, 3330-1, 3340, 3350, 3375 and 3380. The TAPE specification for *devicetype* sets values for the 2400 and 3400 device types.

The actual block size used is consistent with the record format and logical record length of the file, if that information is known or specified.

The following can be specified for *value*:

number
specifies the block size that the SAS System is to use for the device.

OPT
specifies that the SAS System is to choose an optimum block size for the device.

MAX or FULL
specifies that the SAS System is to use the maximum permitted block size for the device.

HALF, THIRD, FOURTH, or FIFTH
specifies that the SAS System is to use the largest value that results in obtaining two, three, four, and five, respectively, blocks per track (if a disk device) or the maximum permitted block size divided by two, three, four, and five, respectively (if not a disk device).

The following example directs the SAS System to use a maximum block size of 6400 for all external files on all disk and tape devices, except for 3350s where a value of 6160 is to be used:

```
OPTIONS FILEBLKSIZE(DISK)=6400
        FILEBLKSIZE(TAPE)=6400
        FILEBLKSIZE(3350)=6160 ;
```

Operating systems: CMS, OS, VM/PC, and VSE

FILLMEM='*hexvalue*
specifies the four hexadecimal characters to be used by the MEMFILL option. This option is for debugging purposes.

Operating systems: CMS, OS, VM/PC, and VSE

FILSZ/NOFILSZ
specifies whether the system sort utility supports the FILSZ parameter. If the IBM ICEMAN (SM01), SyncSort™ * or CA-SORT™ program product sort utility (or equivalent) is installed and supports the FILSZ parameter, specify the FILSZ option to increase the sort efficiency. This option is ignored if you use SORTPGM=SAS.

Operating systems: CMS, OS, and VM/PC

FIRSTOBS=*n*
specifies the number of the first observation that the SAS System is to process. Normally the SAS System begins with the first observation in a data set. For example, if FIRSTOBS=50, the SAS System begins processing with the 50th observation of the data set.

Note that this option applies to every data set used in a job. Thus, in this example,

```
OPTIONS FIRSTOBS=11;
DATA A;
   SET OLD; /* 100 OBSERVATIONS */
DATA B;
   SET A;
DATA C;
   SET B;
```

* SyncSort is a trademark of SyncSort Incorporated.

data set OLD has 100 observations, data set A has 90, B has 80, and C has 70. To avoid decreasing the number of observations in successive data sets, reset FIRSTOBS=1 at an appropriate point in your SAS statements.

If the SAS System is processing a file of raw data, the FIRSTOBS option specifies the first card or line of data that the SAS System should process.

The data set and INFILE statement option FIRSTOBS= take precedence over the FIRSTOBS= system option. (See "SAS Files" for a discussion of the FIRSTOBS= data set option.)

Operating systems: All

FMTERR/NOFMTERR
controls whether the SAS System indicates an error if it cannot find a format associated with a variable. When NOFMTERR is in effect, the SAS System processes the variable using a default format (usually *w.* or $*w.*). When FMTERR is specified, missing formats are considered errors.

Operating systems: All

FORMCHAR
(*printdevice*)=('*formatting characters*')
specifies the output *formatting characters* for *printdevice*. Formatting characters are used to construct tabular output outlines and dividers. This option is effective only in conjunction with the PRINTDEVICE= option below.

The value given for *formatting characters* is any string or list of strings of characters up to 64 bytes long. If fewer than 64 bytes are specified, the value is padded with blanks on the right. The first eleven characters define the two bar characters, vertical and horizontal, and the nine corner characters: upper left, upper middle, upper right, middle left, middle middle (cross), middle right, lower left, lower middle, and lower right.

The standard values for these characters are the following:

```
|----|+|---+=|-/\<>*
```

In addition to these eleven characters, nine other characters are used:

Character	Use
12	starting or ending character for an event line
13	fill character for an event line
14	no special use
15	no special use
16	separation character for variables in event line label
17	no special use
18	left arrow for continuing an event line
19	right arrow for continuing an event line
20	highlighting character for holiday name

You can substitute any character or hexadecimal string to customize the table. If standard printers at your installation do not include the vertical bar (|), you can substitute the capital letter I for each vertical bar in the standard string of characters. Note: if you change any value for a character with the OPTIONS statement, you must specify all the characters. You can use a FORMCHAR option on the FREQ, TABULATE, and CALENDAR procedures if you want to change a single character value.

Specifying all blanks,

```
FORMCHAR(STANDARD) = (' ')
```

produces output with no outlines or dividers. If you are routing your printed output to an IBM 6670 using an extended font (typestyle 27 or 225) with input character set 216, we recommend

```
FORMCHAR(IBM6670)=
   ('FABFACCCBCEB8FECABCBBB4E7E4F6061E04C6EAF'X)
```

If you are using a printer with the standard "text" graphics (for example, a TN print train on an IBM 1403, 3211, or 3203; or a text character set on an IBM 3800), we recommend

```
FORMCHAR(TEXT)=
   ('4FBFACBFBC4F8F4FABBFBB4E7E4F6061E04C6EAF'X)
```

See "SAS Log and Procedure Output" for more detailed information on these characters.

Operating systems: CMS, OS, VM/PC, and VSE

FORTG
FORTG=*filename*
: names a TXTLIB to be searched when the GLOBAL command is issued. This option can be used for user-written procedures or other programs called from SAS programs.

Operating systems: CMS and VM/PC

FS/NOFS
: specifies whether certain SAS facilities and procedures are to operate in full-screen mode when the SAS System is executing interactively on a full-screen terminal. The FS option controls all full-screen facilities **except** the SAS Display Manager System. To invoke display manager, you must use the DMS option.

Operating systems: All

FSDEVICE= *device*
FSD= *device*
: specifies the name of the device you are using when you invoke the SAS display manager. This option can be specified at SAS invocation, or it can be entered in response to the SAS prompt when the DMS option is specified.

Operating systems: AOS/VS, PRIMOS, and VMS

FSP/NOFSP
: specifies that SAS full-screen terminal support and the SAS/FSP facilities are to be available. If you specify NOFSP, SAS execution

requires slightly less memory, but you cannot run any SAS/FSP procedures or use the SAS HELP facility in full-screen mode. It is not necessary for you to have the SAS/FSP software product installed in order to specify this option.

Operating systems: CMS, OS, VM/PC, and VSE

GEN=*n*

specifies the number of generations of history records to save for each SAS data set. The CONTENTS procedure prints the saved generations of historical information for each data set. The GEN= value can range from 0 to 1000.

Operating systems: CMS, OS, VM/PC, and VSE

ICCF/NOICCF

controls the format of the SAS log and whether the SAS System notes the memory and CPU time used for each step. When the ICCF option is in effect, the resource utilization notes are suppressed. When the NOICCF option is in effect, the notes are printed.

Operating system: VSE

IMPLMAC/NOIMPLMAC

controls whether macros defined as statement-style macros can be invoked with statement-style macro calls, as in

```
name parametervalue1 parametervalue2;
```

or if the call must be a name-style macro call, as follows:

```
%name (parametervalue1, parametervalue2)
```

If IMPLMAC is in effect, the macro processor examines the first word of every statement to see if that word is a statement-style macro call. If NOIMPLMAC is in effect, statement-style macro calls are not recognized. If the macro processor encounters a statement-style macro call when NOIMPLMAC is in effect, it treats the call as a SAS statement. The SAS compiler then produces an error message if the statement is not valid or is not used correctly. Regardless of which option is in effect, you can call any macro, including those defined as statement-style macros, with a name-style invocation.

Operating systems: CMS, OS, VM/PC, and VSE

IMS/NOIMS

specifies that the facilities of SAS/IMS-DL/I software are to be available. If the SAS/IMS-DL/I software product is installed and you specify NOIMS, you cannot access IMS or DL/I data bases. The option is ignored if the SAS/IMS-DL/I software product is not installed.

Operating systems: OS and VSE

INCLUDE/NOINCLUDE

controls whether the %INCLUDE facility is available.[8]

Operating systems: All

INITSTMT='*statement*'

specifies a SAS statement or statements to be executed prior to any SAS statements from the SYSIN file (see the SYSIN= option below).

Operating systems: OS and VSE

INVALIDDATA='*character*'
specifies the value that the SAS System is to assign to a variable when invalid data are encountered with an input format (such as in an INPUT statement or the INPUT function). The value specified can be a letter (A-Z, a-z), period (.), or underscore (_).

Operating systems: All

LABEL/NOLABEL
controls whether variable labels are available to SAS procedures. LABEL must be in effect before the LABEL option of any procedure can be used. If NOLABEL is specified, the LABEL option of a procedure is ignored.

Operating systems: All

LAST=*logicalname*
specifies the _LAST_ data set name. After a SAS data set has been created in a SAS job, the value of _LAST_ is the name of the most recently created SAS data set.

Operating systems: All

LDISK/NOLDISK
LD/NOLD
writes the SAS log to a disk file. If this option is specified in an interactive session, the SAS System creates the disk file in addition to displaying the log output on the terminal.[9]

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

LEAVE=*n* or *n*K
Leave *n* or *n*K bytes of memory unallocated. If the SAS System abnormally terminates because of insufficient memory, increase the value of LEAVE. A value specified without a K is not multiplied by 1024; LEAVE=2048 and LEAVE=2K are equivalent.[10]

Operating systems: CMS, OS, VM/PC, and VSE

LINESIZE=*width*
LS=*width*
specifies the printer line width for the SAS log and the standard SAS print file used by the DATA step and procedures. LINESIZE values can range from 64 to 256. When using the SAS System interactively, the TLINESIZE option controls line width for output directed to the terminal, but LINESIZE controls line width for print files.

Operating systems: All

LOG=*fileref*
specifies the SAS log file fileref. This option is reset when you specify the UNITS= option below.

Operating systems: CMS, OS, VM/PC, and VSE

LPRINT/NOLPRINT
LP/NOLP
sends the SAS log to the virtual (default) printer. The LPRINT option has no effect under display manager.[11]

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

LTYPE/NOLTYPE
LT/NOLT
: displays the SAS log on the terminal. LTYPE is the default for interactive, line-mode jobs, and has no effect under display manager.[12]

  Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

MACRO/NOMACRO
: specifies whether or not the SAS macro language, as well as the SYMGET and SYMPUT functions, is available. If you specify NOMACRO, SAS execution requires slightly less memory, but you cannot use any macro language facilities.

  Operating systems: CMS, OS, VM/PC, and VSE

MACROGEN/NOMACROGEN
: specifies whether statements generated by macros (either from the SAS macro language or the MACRO statement) are to be printed. The NOMPRINT option must also be specified to use this option.

  Operating systems: CMS, OS, VM/PC, and VSE

MCOMPILE/NOMCOMPILE
: specifies whether the compiler for the SAS macro language is to be loaded by default. The macro language compiler is loaded any time a macro definition is encountered in the SAS program. If you frequently define (or redefine) macros, then specifying MCOMPILE can reduce overhead.

  Operating systems: OS and VSE

MEMERR/NOMEMERR
: controls whether a diagnostic program check abend occurs when the SAS System would ordinarily give a memory-exceeded message. If MEMERR is in effect, the SAS System abends. This option is intended only for checking programs that exceed memory usage limitations.

  Operating systems: CMS, OS, VM/PC, and VSE

MEMFILL/NOMEMFILL
: specifies that memory obtained by allocation and deallocation is filled with the first half of the FILLMEM characters. This option is for debugging purposes.

  Operating systems: CMS, OS, VM/PC, and VSE

MEMRPT/NOMEMRPT
: controls whether the memory usage report is displayed for each step.

  Operating systems: CMS, OS, VM/PC, and VSE

MERROR/NOMERROR
: controls whether the SAS macro language produces a warning message if the macro processor cannot match a macro-like name (a SAS name prefixed with a percent sign) to an appropriate macro keyword. This error condition occurs when a macro keyword (including a macro call) is misspelled, or when a macro is called before being defined, but it can also occur for legitimate SAS names containing percent signs. Specify NOMERROR if your job contains percent signs in words that could be confused with macro keywords.

  Operating systems: CMS, OS, VM/PC, and VSE

MISSING='*character*'
specifies the character to be printed for missing numeric variable values.

Operating systems: All

MLEAVE=*n*
specifies the portion of memory included in the MSIZE= option below to leave unallocated for use by the macro language processor. The value specified for MLEAVE must be at least 2K less than the value specified for MSIZE.

Operating systems: CMS, OS, VM/PC, and VSE

MLOGIC/NOMLOGIC
specifies whether or not the macro language processor is to trace its execution. If MLOGIC is specified, trace information is printed in the SAS log. This option is for diagnostic use and can be used only when NOMPRINT is in effect.

Operating systems: CMS, OS, VM/PC, and VSE

MODECHARS='*characters*'
specifies characters that the SAS System uses for prompts at a terminal in various submodes when the SAS System is executing interactively. The first character prompts for primary input, that is, SAS statements. The second character is the prompt for data following a CARDS statement and for PROC EDITOR data or any other interactive procedure to indicate interactive submode. The third character is used when %INCLUDE statements execute.

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, and VMS

MPRINT/NOMPRINT
controls whether the statements produced by macro execution are printed (in a synthetic, compressed form). If MPRINT is in effect, statements are printed in the form the SAS compiler receives them. For reading ease, each statement begins at the left margin of a new line and with one space between words. When the MPRINT option is in effect, the output normally produced by the MACROGEN, SYMBOLGEN, and MLOGIC options during macro execution is suppressed. (You can, however, specify both MPRINT and SYMBOLGEN, and SYMBOLGEN will remain in effect for macro program statements and macro variable references used outside of macros.) For most applications, MPRINT is the preferred value for this option.

Operating systems: CMS, OS, VM/PC, and VSE

MSIZE=*n*
specifies the amount of memory available to the macro language processor. The value specified for MSIZE must be at least 2K more than the value specified for MLEAVE.

Operating systems: CMS, OS, VM/PC, and VSE

MSYMSIZE=*n*
specifies the initial size of each macro language processor symbol table.

Operating systems: CMS, OS, VM/PC, and VSE

MWORK=*n*
: specifies the size of the work area available to the macro language processor.

  Operating systems: CMS, OS, VM/PC, and VSE

NAME=*filename*
NA=*filename*
: specifies the filename to be assigned to the disk files for printed procedure output (filetype LISTING), the SAS log (filetype SASLOG), and lines entered at the terminal in interactive mode (filetype SAS).

  Operating systems: CMS and VM/PC

NDSVOLS=*nnnnnn*
: specifies the volume serial that causes the SAS System to treat the SAS data set reference as if you had specified _NULL_. When the SAS System determines that the control language references this volume, the SAS data set is treated as though _NULL_ had been specified. This option is useful for production SAS job streams that, for example, need to initialize catalog generation data groups.

  Operating system: OS

NEWS/NONEWS
: controls whether the text identified by the SASNEWS= option (see below) in the SAS HELP library is to be printed on the SAS log or displayed on the terminal at the beginning of each SAS execution. If NEWS is in effect, the installation-maintained news information is printed at the beginning of every SAS job or at the start of an interactive SAS session.

  Operating systems: All

NOTES/NONOTES
: controls whether notes are printed on the SAS log. (These messages usually begin with NOTE:.) If the NONOTES option is specified, informative messages are not printed on the SAS log. NOTES must be specified on SAS jobs that are sent to SAS Institute for problem determination and resolution; SAS Institute cannot help with program or system problems if notes are not printed.

  Operating systems: All

NULLEOF/NONULLEOF
: determines whether or not to allow a null line to signal the end of the SAS source file in interactive mode. A line beginning with /* always signals the end of the SAS source, whether or not NULLEOF is specified.

  Operating systems: CMS and VM/PC

NUMBER/NONUMBER
: controls whether the page number prints on the first title line of each SAS printed output page.

  Operating systems: All

OBS= *n*
OBS= *MAX*
specifies the **last** observation that the SAS System is to process from a data set. Normally, the SAS System processes all the observations in a data set. For testing purposes you can select the first *n* observations of each SAS data set or the first *n* lines of a raw data file with the OBS= option.

For example, specifying

```
OPTIONS OBS=50;
```

as the first statement of a SAS job causes the SAS System to read only through the 50th observation when reading a SAS data set. Analysis of SAS data sets in PROC steps is also controlled by the OBS= option, so in this example, only the first 50 observations of any data set would be analyzed in a subsequent PROC step.

If the SAS System is processing a file of raw data, the OBS= option specifies how many cards or lines of data to read. SAS counts a line of input data as one observation even if the raw data for several SAS data set observations are on a single line.

You can check the syntax of SAS statements in a job by specifying

```
OPTIONS OBS=0 NOREPLACE;
```

as the first statement in a job. However, since the SAS System actually executes each DATA and PROC step in the job (using no observations), SAS can take certain actions even when the OBS=0 and NOREPLACE options are in effect. For example, SAS executes procedures that process the directories of SAS data sets (such as CONTENTS, DELETE, and DATASETS), and PROC RELEASE. External files are also opened and closed. Thus, even if you specify OBS=0 and your job writes to an external file with a PUT statement, an end-of-file mark is written and any existing data in the file are deleted.

You can use the FIRSTOBS= and OBS= options together to process a set of observations from the middle of a data set. For example, to process only observations 1000 through 1100, specify

```
OPTIONS FIRSTOBS=1000 OBS=1100;
```

The OBS= data set and INFILE statement option takes precedence over the OBS= system option. See "SAS Files" for a discussion of the OBS= data set option.

Operating systems: All

OFFLINE=*n.nnn*
gives the offline storage cost. The cost of storing an offline data set is computed from this option and printed by the CONTENTS procedure. The OFFLINE value is in units of dollars per track per day. Specifying OFFLINE=0 suppresses this feature.

Operating systems: OS and VSE

ONLINE=*n.nnn*
gives the on-line cost for disk storage. The cost of storing an on-line data set is computed from this option and is printed by the CONTENTS procedure. The ONLINE value is in units of dollars per track per day. Specifying ONLINE=0 suppresses this feature.

Operating systems: OS and VSE

OPLIST/NOOPLIST
: specifies whether options given at SAS invocation are printed at the beginning of the SAS log.

  Operating systems: CMS, OS, VM/PC, and VSE

OVP/NOOVP
: controls whether SAS printed output lines can be overprinted. For example, when the SAS System encounters an error in a SAS statement, it prints underscores beneath the word in error if OVP is in effect. If NOOVP is in effect, the SAS System prints dashes on the next line below the error. Note: when displaying output on a terminal screen, OVP is overridden and changed to NOOVP.

  Operating systems: All

PAGES=*n*
PAGES=*MAX*
P=*n*
P=*MAX*
: specifies the maximum number of pages of printed output a SAS job can print. If PAGES=MAX is specified, the SAS System does not limit the number of pages printed. Note: when used in the OPTIONS statement, the value of this option cannot exceed the value in effect (by default or by specifying a value) at SAS invocation. In all cases, local computer installation limits also apply to SAS jobs.

  Operating systems: All

PAGESIZE=*n*
PS=*n*
: specifies the number of lines that can be printed per page of SAS output. If the SAS System is running in an interactive environment, the PAGESIZE= option defaults to the terminal screen size (if this information is available from the operating system). The value specified can be from 20-500.

  Operating systems: All

PARM='*string*'
: specifies a parameter string passed to both the SAS System and external procedures. When a PARM string is specified with this option, the string is passed to the next procedure that the SAS supervisor executes. This facility permits you to pass parameters to user-written procedures. Note: the parameter string does not need to be specified before the PROC statement that specifies the user-written procedure, only before the RUN statement or the next PROC or DATA statement.

  Because the string is also passed to all subsequent SAS procedures, which generally do not support its specification, the option should be reset to a null string (that is: OPTIONS PARM='';) after the external procedure executes and before the next SAS procedure is invoked. For example:

```
PROC program CC=4;
OPTIONS PARM='parm to external procedure';
RUN;
OPTIONS PARM='' /* RESET FOR SAS PROCS */ ;
more SAS statements
```

Operating systems: CMS, OS, VM/PC, and VSE

PARMCARDS=*fileref*
specifies the fileref for the PARMCARDS data set. This option is reset when the UNITS= option, below, is specified. For example, PARMCARDS= FT15F001.

Operating systems: CMS, OS, and VM/PC

PDISK/NOPDISK
PD/NOPD
writes the printed procedure output to a disk file. If this option is specified in an interactive session, the SAS System creates the disk file in addition to displaying the output on a terminal. PDISK is the default value for noninteractive jobs. This option has no effect under display manager.[13]

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

PLIO *filename*
names the PL/I optimizing compiler run-time subroutine library on your system. The PLIO library must be specified to execute SAS procedures compiled with the PL/I optimizing compiler. This option should be specified when SAS is installed.

Operating systems: CMS and VM/PC

PPRINT/NOPPRINT
PP/NOPP
sends the procedure output to the virtual (default) printer. The PPRINT option has no effect under display manager.

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

PRINTDEVICE=*device*
PRINTDEV=*device*
specifies the device description or name of the destination for SAS printed output. This option determines, for example, which set of FORMCHAR (*printdevice*) formatting characters to use. The following names are currently standard:

Name	**Characteristics Assumed**
STANDARD	none
TEXT	text (TN print train) graphics
IBM6670	model 2 with extended fonts
IBM3800	text plus grey scale graphics

Operating systems: CMS, OS, and VM/PC

PRINTHOVP/NOPRINTHOVP
specifies whether or not lines flagged as header lines in SAS procedure output are overprinted in printed output.

Operating systems: AOS/VS, PRIMOS, and VMS

PRINTINIT/NOPRINTINIT
initializes the SAS print file. Specify NOPRINTINIT if a previous program or job step has already written output to the same file, and you want that output preserved.

Operating systems: CMS, OS, and VM/PC

PROBSIG=*n*
controls the formatting of *p*-values in all statistical procedures. When PROBSIG=0, *p*-values are printed with four decimal places and truncated at .0001.

PROBSIG=1 guarantees that *p*-values are printed with at least one significant digit; that is, values greater than .000095 are printed with four decimal places, but values less than .000095 are printed in E-notation.

PROBSIG=2 guarantees at least two significant digits so that values greater than .0000995 are printed with five decimal places, and smaller values are printed in E-notation.

Operating systems: All

PROCSIZE=*maximum*
specifies the maximum number of bytes a procedure can allocate in a single request to the SAS procedure interface routine. You can use this option in special environments (for example, ROSCOE™)* where more than one application or user is executing in the same region or partition. This option is **not** applicable to jobs run under standard IBM TSO and standard IBM ICCF.

Operating systems: OS and VSE

PSEG *value*
determines the use of a CMS saved segment for portions of the SAS System. The following can be specified for *value*:

- OFF specifies that P-level saved segments are not to be used.
- NONSHARED (or NONSHARE) specifies that the P-level saved segments are to be used but no portions shared with other users.
- ON specifies that P-level saved segments are to be used and reentrant portions shared, if possible. If not possible, the P-level segments are not shared.
- SHARED (or SHARE) specifies that the P-level saved segments are to be used and reentrant portions shared among CMS users.

Operating system: CMS

PTYPE/NOPTYPE
PT/NOPT
displays the procedure output on a terminal. PTYPE is the default value for an interactive line-mode session. This option has no effect under display manager.

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

REPLACE/NOREPLACE
controls the replacement of permanently stored SAS data sets. If NOREPLACE is specified, a permanently stored SAS data set cannot be replaced with one of the same name. This prevents the inadvertent replacement of existing SAS data sets.

The REPLACE= data set option takes precedence over the REPLACE/NOREPLACE system option. See "SAS Files" for a discussion of the REPLACE= data set option.

---

* ROSCOE is a trademark of Applied Data Research.

Operating systems: All

S=*n*

specifies the length of statements on each line of source statement and the data on lines following a CARDS statement. When S=0, the SAS System determines whether or not the input is sequence numbered by looking at the sequence field of the first statement. If the entire sequence field of the first statement is numeric, then the SAS System assumes the file is sequence numbered. If the value of S is greater than zero, the SAS System uses that value of S as the length of the data to be scanned and does not look for sequence numbers. S cannot be set to a value less than zero. Currently, primary input to the SAS System must be fixed length records of eighty characters. If sequence numbers are present, they should be in columns 73-80 (refer to the SEQ= option, below, for more information on sequence numbers).

Operating systems: CMS, OS, VM/PC, and VSE

S2=S
S2=*n*

specifies the length of secondary source statements. When S2=S, text included with a %INCLUDE statement uses the current S= value. The S2= option behaves exactly like S= option but controls only input from a %INCLUDE statement. Text included via %INCLUDE may be either fixed or varying length. Fixed length records are either unsequenced or sequenced in the last eight columns, whereas varying length records are either unsequenced or sequenced in the first eight columns. For varying length records, S2 is the starting column of the data rather than the ending column as for fixed length records.

Operating systems: CMS, OS, VM/PC, and VSE

S370/NOS370

specifies that the machine upon which the SAS System is executing supports the IBM System/370 instruction set (for example, a 303X, 43XX, or 308X processor). Specify NOS370 only if you are executing the SAS System on an IBM System/360 and the OS/360 program check first level interrupt handler (FLIH) has been modified to simulate in software the execution of the System/370 nonprivileged instructions.

Operating systems: CMS, OS, VM/PC, and VSE

SASHELP=*fileref*

specifies the fileref of the HELP library. The HELP library is accessed with the HELP statement or by specifying the NEWS option at SAS invocation.

Operating systems: CMS, OS, and VM/PC

SASLIB *filename*

directs SAS to issue a CMS GLOBAL command to add the file

```
filename   TXTLIB   *
```

to the list of TXTLIBs searched during a SAS session. You need to use the SASLIB option only if you have placed user-written formats in a TXTLIB.

The SASLIB file is then automatically searched for user-written formats referenced in your SAS job. See PROC FORMAT for a complete discussion.

Operating systems: CMS and VM/PC

SASNEWS=*member*
specifies which member of the HELP library is printed on the SAS log when the NEWS option is specified at SAS invocation.

Operating systems: All

SEQ=*length*
specifies the length of the numeric portion of the sequence field. When SEQ=8, all eight characters in the sequence field are numeric. The SAS System always assumes an eight-character sequence field; however, some editors place some alphabetic information in the first several characters, such as the file name. The SEQ= value specifies the number of digits that are right justified in the eight-character field. Thus, for the sequence field AAA00010, specify SEQ=5. In this case, the SAS System looks at only the last five characters of the eight-character sequence field, and, if numeric, it assumes that the whole eight-character field is a sequence field.

Operating systems: CMS, OS, VM/PC, and VSE

SERIES *value*
specifies which series of discontiguous saved segments should be used. When *value* is a single letter, it specifies which series of saved segments are to be used for both the S- and P-levels. When *value* is two letters, the first letter specifies which series of saved segments are to be used for the S-level. The second letter specifies which series of saved segments are to be used for the P-level.

Operating system: CMS

SERROR/NOSERROR
controls whether a warning message is produced when the SAS System encounters a macro variable (also called a symbolic variable) reference that the macro processor cannot match with an appropriate macro variable. This error condition occurs when the name in a macro variable reference is misspelled or if the variable is referenced before being defined, or if the job contains ampersands (&) followed by SAS names without an intervening blank. Specify NOSERROR if ampersands in your job could be interpreted as macro variable references (for example, if the ampersand is a symbol for the logical operator AND with no intervening blanks, as in 'X=Y&Z').

Operating systems: CMS, OS, VM/PC, and VSE

SIODISK *modeletter*
forces all temporary files allocated by the CMS SAS interface to a particuliar CMS minidisk. If the SIODISK option is not specified, the SAS System allocates temporary files on the write-disk with the most available free space.

Operating systems: CMS and VM/PC

SKIP=*n*
specifies the number of lines to skip at the top of each page of SAS output before printing the first title line. The location of the first line

is relative to the position established by the carriage control tape or forms control buffer on the printer. Most installations define this so that the first line of a new page (top of form) begins three or four lines down the form. If this spacing is sufficient, specify SKIP=0 so that additional lines are not skipped. The SKIP value can range from 0 to 20. The SKIP value does not affect the maximum number of lines that are printed on each page.

Operating systems: All

SNP/NOSNP
SNP specifies that internal SAS system information is to be dumped in hexadecimal form as a diagnostic trace of SAS activity. This option is for diagnostic use.

Operating systems: CMS, OS, and VM/PC

SNPPROG/NOSNPPROG
SNPPROG specifies that the compiled SAS code is to be dumped on the SAS log. If you specify SNPPROG, then SNP (see above) is assumed. This option is for diagnostic use.

Operating systems: CMS, OS, and VM/PC

SORT=*n*
specifies the minimum size in cylinders of the temporary data sets that PROC SORT dynamically allocates for sort work areas. The SORT= option is ignored if the DYNALLOC option is specified.

Operating system: OS

SORTDEV=*device*
specifies the device (unit) name used under MVS to allocate the sort work data sets dynamically. The value specified must be a generic or site-dependent name defined at MVS system generation for real, non-VIO, direct access devices.

Operating system: OS

SORTLIB='*datasetname*'
specifies the name of the sort library. This option may be restricted by your installation.[14]

Operating systems: CMS, OS, and VM/PC

SORTLIST/NOSORTLIST
controls whether you can request additional information about the system sort when PROC SORT is invoked. If the SORTLIST option is specified, the SAS System passes the LIST parameter to the system sort utility. If NOSORTLIST is specified, the SAS System does not pass the LIST parameter to the system sort utility.

Operating systems: CMS, OS, VM/PC, and VSE

SORTMSG/NOSORTMSG
specifies whether or not the MSG=AP option is to be passed to the system sort utility. Specifying SORTMSG is equivalent to specifying the MESSAGE option in each PROC SORT statement.

Operating systems: CMS, OS, and VM/PC

SORTMSG=*fileref*
specifies the fileref to be dynamically allocated on MVS systems by PROC SORT for the system sort utility message print file. Under TSO,

this file is allocated to the terminal. This option may be restricted by your installation.

Operating system: OS

SORTPGM='*utility*'
specifies the name of the system sort utility to be invoked by the SAS System. If SORTPGM=SAS is specified, then a SAS-supplied sort utility is invoked that is suitable for sorting data sets with a small number of observations and variables.[15] This option may be restricted by your installation.

Operating systems: CMS, OS, VM/PC, VMS, and VSE

SORTSIZE=MAX
SORTSIZE=SIZE
SORTSIZE=*n*
SORTSIZE=*n*K
specifies the SIZE parameter to be passed to the system sort utility.[16]

Operating systems: CMS, OS, VM/PC, and VSE

SORTWKDD='*prefix*'
specifies the prefix of the sort work area filerefs to be dynamically allocated by either the SAS System (when NODYNALLOC is specified) or the system sort utility (when DYNALLOC is specified). For example, if you specify SORTWKDD='XYPG', filerefs XYPGWK01, XYPGWK02, and so forth, are dynamically allocated. This option is ignored except under MVS. This option may be restricted by your installation.

Operating system: OS

SORTWKNO=*n*
specifies the minimum number of sort work areas to be allocated. Refer to the PROC SORT chapter for more information on the use of the SORTWKNO options. This option may be restricted by your installation.[17]

Operating systems: OS and VSE

SOURCE/NOSOURCE
controls whether SAS source statements are printed on the SAS log.

Operating systems: All

SOURCE2/NOSOURCE2
controls whether secondary source statements from files included by %INCLUDE are printed on the SAS log.

Operating systems: All

SPOOL/NOSPOOL
controls whether SAS statements are spooled to a utility data set in the WORK data library. While the SPOOL option is in effect, SAS statements are saved for later use by a %INCLUDE statement. While the NOSPOOL option is in effect, the SAS statements are not saved. Switching from NOSPOOL to SPOOL erases any previously saved statements. If you switch from SPOOL to NOSPOOL, however, you can still include the saved statements as long as NOSPOOL is in effect.

Operating systems: CMS, OS, VM/PC, and VSE

SSEG SHARED
SSEG NONSHARED
SSEG OFF
SSEG ON
determines the use of a CMS saved segment for portions of the SAS supervisor and associated library routines. If your CMS installation has installed a CMS SAS saved supervisor segment, memory requirements for SAS DATA and PROC steps are reduced.

Note that saved segments are installed for certain sizes of virtual machines. If you increase your virtual machine size above the segment's load point, the saved segment is not used. Check with your CMS support personnel for appropriate virtual machine sizes to use with CMS SAS saved supervisor segments.

SHARED specifies that the saved supervisor segment is to be used and reentrant portions are to be shared among CMS SAS users.

NONSHARED specifies that the saved supervisor segment is to be used but no portions are to be shared with other users. Use NONSHARED only as a temporary bypass if some of the shared portions turn out not to be reentrant.

OFF specifies that the SAS System is to execute from the modules on the CMS SAS disk; the saved supervisor segment is not to be used.

ON specifies that the saved supervisor segment is to be used.

If the SAS System cannot use the saved supervisor segment, either because it has not been installed at your installation or because of the virtual machine size, the status of the SSEG option is set to NOT_USED.

Operating system: CMS

STIMER/NOSTIMER
specifies whether SAS notes the computer resources used for each step on the SAS log.[18]

Operating systems: AOS/VS, PRIMOS, OS, VMS, and VSE

SVCHND/NOSVCHND
determines if SAS supervisor call handling is in effect when executing a CMS statement from within a SAS session.

Operating systems: CMS and VM/PC

SYMBOLGEN/NOSYMBOLGEN
controls whether the text produced by expanding macro variable (also called symbolic variable) references is printed on the SAS log. If SYMBOLGEN is specified, the values to be substituted for each macro variable are listed on the SAS log. If the MPRINT option is also specified, SYMBOLGEN remains in effect for macro program statements and macro variable references only when they are used outside of macros.

Operating systems: CMS, OS, VM/PC, and VSE

SYNCSORT/NOSYNCSORT
specifies that the SyncSort™ program product is used as the system sort utility.

Operating systems: OS and VSE

SYSIN=*fileref*
specifies the fileref for the primary SAS input stream data set. The fileref for the SAS input stream is normally SYSIN, but it can be changed to another fileref with this option. If you are executing the SAS System in an interactive environment, and you do **not** explicitly specify a SYSIN= value at SAS invocation, then the SAS System requests input directly from the terminal.[19]

Operating systems: OS and VSE

SYSIN/NOSYSIN
specifies whether or not the SAS System is to read the primary SAS input stream from the SYSIN= fileref. NOSYSIN specifies that the SYSIN= fileref is not to be used. In this case, you must use the INITSTMT= option (described above) to provide a primary input stream to the SAS System.

Operating systems: OS and VSE

SYSPARM='*characters*'
specifies a character string that can be passed to SAS programs. The character string specified can be accessed in a SAS DATA step by the SYSPARM() function (see "SAS Functions") or at any time during the job by using the automatic macro variable reference &SYSPARM (see "The SAS Macro Language"). The maximum length for SYSPARM is 200 characters.[20]

Operating systems: CMS, OS, VM/PC, and VSE

TAPE=*device*
gives the site-dependent or generic name defined at OS system generation for tape devices.

Operating system: OS

TAPECLOSE=REREAD
TAPECLOSE=LEAVE
TAPECLOSE=REWIND
TAPECLOSE=DISP
specifies the default CLOSE disposition (volume positioning) to be performed when a SAS data library on tape is closed. The default value for this option is REREAD, except for tape libraries accessed by PROC COPY. The only valid value for tape libraries accessed by PROC COPY is LEAVE; it cannot be overridden. The FILECLOSE= data set option takes precedence over the TAPECLOSE= system option. See "SAS Files" for a discussion of the FILECLOSE= data set option.

Specify TAPECLOSE=REREAD if you are accessing one or more tape data libraries several times in a SAS job.[21] The SAS System leaves the tape volume positioned at the beginning of the tape data library. REREAD overrides a FREE=CLOSE specification in the control language.

Specify TAPECLOSE=REWIND if you are not repeatedly accessing one or more tape libraries in a SAS job. The SAS System rewinds the tape volume to the beginning. A FREE=CLOSE specification in the control language overrides the REWIND specification.

Specify TAPECLOSE=LEAVE if you are not repeatedly accessing the same tape libraries in a SAS job, but, once used, you are creating

or accessing one or more tape libraries in a subsequent file on the same tape volume. The SAS System leaves the tape positioned at the end of the data library. LEAVE overrides a FREE=CLOSE specification in the job control.

When you specify TAPECLOSE=DISP, the tape volume is positioned as determined by the operating system, according to specifications in the control language.

Operating systems: AOS/VS, OS, PRIMOS, VMS, and VSE

TEXT82/NOTEXT82

specifies whether the SAS System issues an error message or a warning message when it encounters a statement that requires quotation marks around the value for the statement. For example, when you use the TITLE statement, the title must be enclosed in quotation marks unless TEXT82 is in effect. The SAS System processes a title without quotation marks with the TEXT82 option but issues a warning message indicating that this format is the old style and will not be supported in the future. Note: whether double quotation marks can be used depends on the value specified for the DQUOTE option.

Operating systems: CMS, OS, VM/PC, and VSE

TIME=MAX
T=MAX
TIME=*n*
T=*n*

gives the maximum time in seconds allowed for a SAS job. The value must be an integer (or MAX) and cannot exceed the default value or the value specified at SAS invocation. The SAS System terminates abnormally when the time required for the SAS job exceeds the specified value. If TIME=MAX, the SAS System does not limit the time used. In all cases, local computer installation limits also apply to SAS jobs. [22]

Operating systems: OS and VSE

TLINESIZE=*n*
TLS=*n*

specifies the linesize for displaying SAS output on a terminal. If TLINESIZE=0 when SAS is executing interactively, the default linesize for your terminal determines the linesize of SAS output. The value of the TLINESIZE= option can range from 64 to 132. The LINESIZE= option sets the linesize for all files that are not directed to a terminal.

Operating systems: AOS/VS, CMS, OS, PRIMOS, VM/PC, and VMS

TLOG/NOTLOG
TL/NOTL

determines whether or not the SAS statements entered at the terminal are written to a disk file. TLOG is recognized only when the SAS System is invoked interactively, but has no effect under display manager.[23]

Operating systems: AOS/VS, CMS, PRIMOS, VM/PC, and VMS

TMSGLEV=A
TMSG=NOTES
TMSG=NOTE
TMSG=ERRORS
TMSG=ERRO
TMSG=OFF

determines what informative messages are displayed on the terminal when the entire SAS log is not directed to the terminal.

NOTES specifies that all notes and errors are displayed.
ERRORS specifies that only error messages are displayed.
OFF suppresses the display of any informative messages.

Operating systems: CMS and VM/PC

TPAGESIZE=*n*
TPS=*n*

specifies the number of lines of SAS output displayed on a terminal screen. The default value is 0. The value is derived from the terminal screen size and can range from 20 to 500. Note: see the PAGESIZE= option.

Operating systems: All

TSO/NOTSO

under TSO, controls the format of the SAS log and whether the SAS System notes the memory and CPU time used for each step. If the TSO option is in effect, the resource utilization notes are suppressed; if the NOTSO option is in effect, the notes are printed.

Operating systems: CMS, OS, and VM/PC

TXTLIB/NOTXTLIB

determines whether or not TXTLIBs included in a CMS GLOBAL TXTLIB command issued before SAS invocation are searched during a SAS session. If TXTLIB is specified, the TXTLIBs included in a CMS GLOBAL TXTLIB specification are added to the end of the list of libraries searched during a SAS session. With either setting of this option, the list of GLOBALed TXTLIBs is the same at the end of the SAS session as it was before SAS invocation.

Operating systems: CMS and VM/PC

UNITS=11 12 13 14 15 16 17 18 19 20

controls units (fileref) assignment.

To eliminate possible conflicts with FORTRAN and other programs, these unit numbers are usually defined with the UNITS= option to the values below:

11 =	Log file
12 =	Print file
13 =	Punch file
14 =	Plot file
15 =	PARMCARDS file
16-20=	Undefined

The SAS cataloged procedures and CLISTs reference these unit numbers. The keyword UNITS must be followed by a list of up to ten numbers, separated by blanks. The first number given becomes the unit number for log output; the second for procedure output; and so on.

Specifying the UNITS= option resets the PARMCARDS= and LOG= options. If both UNITS= and LOG= or if PARMCARDS= is to be specified, UNITS= should precede the LOG= or PARMCARDS= specification.[24]

Operating systems: OS and VSE

USER=*name*

specifies the first-level name that is automatically assumed for all one-level SAS data set names in a SAS program. When the USER= option is specified, the SAS System uses the name of the permanent SAS data set specified in the USER= option as the first-level name.[25]

When USER=WORK (the default) is in effect, SAS assumes that one-level names refer to temporary WORK data sets.

Operating systems: All

USERPARM='*data*'

specifies up to 200 bytes of data to be made available to SAS installation exits.

Operating systems: CMS, OS, VM/PC, and VSE

VIOBUF=*n*

specifies the maximum size (in kilobytes) of virtual I/O storage allocated for the SAS WORK data library. The VIOBUF= option does not allocate a fixed amount of space, but rather sets a limit on the amount of space.

Operating systems: CMS and VM/PC

VNFERR/NOVNFERR

prevents the SAS System from setting the error flag for a missing variable when a _NULL_ data set (or a data set that is bypassed by the operating system control language) is used in a MERGE statement of a DATA step. When NOVNFERR is specified, the SAS System still issues a warning for a variable not found (VNF), but it does not tally an error or stop processing.

Operating systems: CMS, OS, VM/PC, and VSE

VSAMLOAD/NOVSAMLOAD

controls whether you can load VSAM files. If your installation does not establish VSAMLOAD as the default, you must specify VSAMLOAD before attempting to use a FILE statement without an associated INFILE statement to load a VSAM file.

Operating systems: CMS, OS, and VSE

VSAMREAD/NOVSAMREAD

controls read access to VSAM files. If your installation does not establish VSAMREAD as the default, you must specify VSAMREAD before attempting to read records in a VSAM file.

Operating systems: CMS, OS, and VSE

VSAMUPDATE/NOVSAMUPDATE
controls update access to VSAM files. If your installation does not establish VSAMUPDATE as the default, you must specify VSAMUPDATE before attempting to update, erase, or replace records in a VSAM file. To reload a VSAM file you must specify VSAMLOAD, described above.

Operating systems: CMS, OS, and VSE

WORK=*name*
gives the libref for the temporary SAS files. If WORK= is not specified, the libref of temporary SAS files is WORK.[26]

Operating systems: All

WORKINIT/NOWORKINIT
controls whether the WORK data library is initialized.

Operating systems: CMS, OS, VM/PC, and VSE

ZEROMEM/NOZEROMEM
controls whether all memory is set to binary zeros. This option is for diagnostic use only and should be used only when requested by a SAS consultant.

Operating systems: CMS and VM/PC

OPTIONS FOR SPECIFIC OPERATING SYSTEMS

AOS/VS, PRIMOS, and VMS Operating Systems

Table 18.2 Options for the AOS/VS, PRIMOS, and VMS Operating Systems

Option	Default Value	SAS Invocation	OPTIONS Statement
BATCH/INTERACTIVE	see **Table 18.1**	x	x
C60/C48/C96	C96	x	x
CAPS	NOCAPS	x	x
CENTER	CENTER	x	x
CHARCODE	NOCHARCODE	x	x
DATE	DATE	x	x
DMS	NODMS	x	x

continued on next page

Table 18.2 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
DQUOTE	DQUOTE	x	x
ERRORABEND	NOERRORABEND	x	x
ERRORS=	20	x	x
FIRSTOBS=	1	x	x
FMTERR	FMTERR	x	x
FS	NOFS	x	x
FSD=	no default	x	x
INCLUDE	INCLUDE	x	x
INVALIDDATA=	.	x	x
LABEL	LABEL	x	x
LAST	_NULL_	x	x
LDISK	Batch:LDISK Interactive: NOLDISK Noninteractive: LDISK	x	x
LINESIZE=	Interactive: 80 Noninteractive: 132 Batch: 132	x	x
LPRINT	NOLPRINT	x	x
LTYPE	Batch: NOLTYPE Interactive: LTYPE Noninteractive: NOLTYPE	x	x
MISSING=	.	x	x
MODECHARS=	? > *	x	x
NEWS	NONEWS	x	x
NOTES	NOTES	x	x
NUMBER	NUMBER	x	x
OBS=	2147483647	x	x

continued on next page

Table 18.2 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
OVP	Interactive: NOOVP Noninteractive: OVP BATCH: OVP	x	x
PAGES=	2147483647	x	x
PAGESIZE=	Interactive: 24 Noninteractive: 60 Batch: 60	x	x
PDISK	Interactive: NOPDISK Noninteractive: PDISK Batch: PDISK	x	x
PPRINT	NOPPRINT	x	x
PRINTHOVP	NOPRINTHOVP	x	x
PROBSIG=	0	x	x
PTYPE	Interactive: PTYPE Noninteractive: NOPTYPE Batch: NOPTYPE	x	x
REPLACE	REPLACE	x	x
SASNEWS=	SASNEWS	x	x
SKIP=	0	x	x
SOURCE	Interactive: NOSOURCE Noninteractive: SOURCE Batch: SOURCE	x	x
SOURCE2	SOURCE2	x	x
STIMER	STIMER	x	x
TAPECLOSE=	no default	x	x
TLINESIZE=	80	x	x
TLOG	NOTLOG	x	x
TPAGESIZE=	0	x	x
USER=	WORK	x	x
WORK=	WORK	x	x

CMS and VM/PC Operating Systems

Table 18.3 Options for the CMS and VM/PC Operating Systems

Option	Default Value	SAS Invocation	OPTIONS Statement
BATCH/INTERACTIVE	see **Table 18.1**	x	
BAUD=	1200	x	x
BLKSIZE=	8192	x	x
BLKSIZE(devicetype)	CMS		x
BYERR	BYERR	x	x
C60/C48/C96	C96	x	x
CAPS	NOCAPS	x	x
CARDS=	MAX	x	x
CENTER	CENTER	x	x
CHARCODE	NOCHARCODE	x	x
CHKPT	NOCHKPT	x	x
CMDMSG	NOCMDMSG	x	x
CMS	Interactive: CMS Noninteractive: NOCMS	x	x
CPSP	CPSP	x	x
DATE	DATE	x	x
DEFAULT	Interactive: CMS Noninteractive: Batch	x	x
DEVICE=	no default	x	x
DMS	Interactive: DMS Noninteractive: NODMS	x	
DQUOTE	DQUOTE	x	x

continued on next page

Table 18.3 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
DSNFERR	DSNFERR	x	
DUMP	NODUMP	x	
DUMPSYS	NODUMPSYS	x	
ERASE	ERASE	x	x
ERRORABEND	NOERRORABEND	x	x
ERRORS=	20	x	x
FILCLR	NOFILCLR	x	x
FILEBLKSIZE(CMS)	32760	x	x
FILLMEM=	'FFFF'X	x	x
FILSZ	FILSZ	x	x
FIRSTOBS=	1	x	x
FMTERR	FMTERR	x	x
FORMCHAR(STANDARD)	|----|+|---+=|-/\<>*	x	x
FORTG	no default	x	x
FS	NOFS	x	
FSD=	no default	x	x
FSP	NOFSP	x	
GEN=	5	x	x
IMPLMAC	IMPLMAC	x	x
INCLUDE	INCLUDE	x	x
INVALIDDATA=		x	x
LABEL	LABEL	x	x
LAST	_NULL_	x	x

continued on next page

Table 18.3 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
LDISK	Interactive: NOLDISK Noninteractive: LDISK	x	
LEAVE	0	x	x
LINESIZE=	Interactive: 80 Noninteractive: 132	x	x
LOG=	FT11F001	x	
LPRINT	NOLPRINT	x	
LTYPE	Interactive: LTYPE Noninteractive: NOLTYPE	x	
MACRO	NOMACRO	x	
MACROGEN	NOMACROGEN	x	x
MEMERR	NOMEMERR	x	x
MEMFILL	NOMEMFILL	x	x
MEMRPT	NOMEMRPT	x	x
MERROR	MERROR	x	x
MISSING=	.	x	x
MLEAVE=	6144	x	
MLOGIC	NOMLOGIC	x	x
MODECHARS=	?>*	x	x
MPRINT	NOMPRINT	x	x
MSIZE=	12228	x	
MYMSIZE=	1024	x	
MWORK=	2048	x	
NAME=	SAS	x	x
NEWS	NONEWS	x	

continued on next page

Table 18.3 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
NOTES	NOTES	x	x
NULLEOF	NONULL	x	x
NUMBER	NUMBER	x	x
OBS=	2147483647	x	x
OPLIST	NOOPLIST	x	
OVP	Interactive: NOOVP Noninteractive: OVP	x	x
PAGES=	2147483647	x	x
PAGESIZE=	60	x	x
PARM=	"	x	x
PARMCARDS=	FT15F001	x	x
PDISK	Interactive: PDISK Noninteractive: PDISK	x	
PLIO	no default	x	
PPRINT	NOPPRINT	x	
PRINTDEVICE=	STANDARD	x	
PRINTINIT	NOPRINTINIT	x	x
PROBSIG=	0	x	x
PSEG=*	SHARED	x	x
PTYPE	NOPTYPE	x	
REPLACE	REPLACE	x	x
S=	0	x	x
S2=	S	x	x
S370	S370	x	
SASHELP=	SASHELP	x	x

Continued

Table 18.3 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
SASLIB	no default	x	
SASNEWS=	SASNEWSC	x	x
SEQ=	8	x	x
SERIES*	A	x	
SERROR	SERROR	x	x
SIODISK	The read/write disk with the most space	x	
SKIP=	0	x	x
SNP	NOSNP	x	
SNPPROG	NOSNPPROG	x	
SORT=	1	x	x
SORTLIB=	no default	x	x
SORTLIST	NOSORTLIST	x	x
SORTMSG	NOSORTMSG	x	x
SORTPGM=	SAS	x	x
SORTSIZE=	MAX	x	x
SOURCE	SOURCE	x	x
SOURCE2	SOURCE2	x	x
SPOOL	NOSPOOL	x	x
SSEG*	SHARED	x	
SVCHND	NOSVCHND	x	x
SYMBOLGEN	NOSYMBOLGEN	x	x
SYSPARM=	''	x	x
TAPECLOSE=*	REREAD	x	x
TEXT82	TEXT82	x	x

continued on next page

Table 18.3 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
TLINESIZE=	80	x	x
TLOG	NOTLOG	x	x
TMSGLEV=	ERRORS	x	x
TPAGESIZE=	0	x	x
TSO	Interactive: TSO Noninteractive: TSO	x	
TXTLIB	TXTLIB	x	
USER=	USER	x	x
USERPARM=	"	x	x
VIOBUF=	256	x	x
VNFERR	VNFERR	x	x
VSAMLOAD*	NOVSAMLOAD	x	x
VSAMREAD*	NOVSAMREAD	x	x
VSAMUPDATE*	NOVSAMUPDATE	x	x
WORK=	WORK	x	
WORKINIT	WORKINIT	x	
ZEROMEM	NOZEROMEM	x	x

*Does not apply to VM/PC

OS Operating System

Table 18.4 Options for the OS Operating System

Option	Default Value	SAS Invocation	OPTIONS Statement
BATCH/INTERACTIVE	see **Table 18.1**	x	
BAUD=	1200	x	x

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
BLDLTABLE	BLDLTABLE	x	x
BLKSIZE=	0	x	x
BLKSIZE(DASD)	THIRD	x	x
BUFNO=	2	x	x
BYERR	BYERR	x	x
C60/C48/C96	C96	x	x
CAPS	NOCAPS	x	x
CARDS=	MAX	x	x
CENTER	CENTER	x	x
CHARCODE	NOCHARCODE	x	x
CHKPT	Batch: NOCHKPT Interactive: CHKPT Noninteractive: NOCHKPT	x	x
CLIST	NOCLIST	x	
DATE	DATE	x	x
DEFAULT	Batch: BATCH Interactive: TSO Noninteractive: BATCH	x	x
DEVICE=	no default	x	x
DISK=	DISK	x	
DMS	Batch: NODMS Interactive: DMS Noninteractive: NODMS	x	
DQUOTE	DQUOTE	x	x
DSNFERR	DSNFERR	x	
DSRESV	DSRESV	x	x
DUMP	NODUMP	x	

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
DUMPM	NODUMPM	x	x
DUMPP	NODUMPP	x	x
DUMPSYS	NODUMPSYS	x	
DYNALLOC	NODYNALLOC	x	x
ERASE	ERASE	x	x
ERRORABEND	NOERRORABEND	x	x
ERRORS=	20	x	x
FILCLR	NOFILCLR	x	x
FILEBLKSIZE(devicetype)	Optimum for the device type	x	x
FILLMEM=	'FFFF'X	x	x
FILSZ	FILSZ	x	x
FIRSTOBS=	1	x	x
FMTERR	FMTERR	x	x
FORMCHAR(printdevice)	|----|+|---+=|-/\<>*	x	x
FS	NOFS	x	
FSD=	no default	x	x
FSP	NOFSP	x	
GEN=	5	x	x
IMPLMAC	IMPLMAC	x	x
IMS	IMS	x	
INCLUDE	INCLUDE	x	x
INITSTMT=	"	x	x
INVALIDDATA=	.	x	x

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
LABEL	LABEL	x	x
LAST	_NULL_	x	x
LEAVE	0	x	x
LINESIZE=	Batch: 132 Interactive: 80 Noninteractive: 132	x	x
LOG=	FT11F001	x	
MACRO	MACRO	x	
MACROGEN	NOMACROGEN	x	x
MCOMPILE	NOMCOMPILE	x	x
MEMERR	NOMEMERR	x	x
MEMFILL	NOMEMFILL	x	x
MEMRPT	MEMRPT	x	x
MERROR	MERROR	x	x
MISSING=	.	x	x
MLEAVE=	6144	x	
MLOGIC	NOMLOGIC	x	x
MODECHARS=	?>*	x	x
MPRINT	NOMPRINT	x	x
MSIZE=	12288	x	
MYMSIZE=	1024	x	
MWORK=	2048	x	
NDSVOLS	no default	x	x
NEWS	NONEWS	x	

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
NOTES	NOTES	x	x
NUMBER	NUMBER	x	x
OBS=	MAX	x	x
OFFLINE=	0.0000	x	x
ONLINE=	0.0000	x	x
OPLIST	NOOPLIST	x	
OVP	Batch: NOOVP Interactive: NOOVP Noninteractive: OVP	x	x
PAGES=	MAX	x	x
PAGESIZE=	Batch: 60 Interactive: 32 Noninteractive: 60	x	x
PARM=	"	x	x
PARMCARDS=	FT15F001	x	x
PRINTDEVICE=	STANDARD	x	
PRINTHOVP	NOPRINTHOVP	x	
PRINTINIT	PRINTINIT	x	x
PROBSIG=	0	x	x
PROCSIZE=	MAX	x	x
REPLACE	REPLACE	x	x
S=	0	x	x
S2=	S	x	x
S370	S370	x	
SASHELP=	SASHELP	x	x
SASNEWS=	SASNEWSB	x	x

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
SEQ=	8	x	x
SERROR	SERROR	x	x
SKIP=	0	x	x
SNP	NOSNP	x	
SNPPROG	NOSNPPROG	x	
SORT=	1	x	x
SORTDEV=	RIO	x	x
SORTLIB=	no default	x	x
SORTLIST	NOSORTLIST	x	x
SORTMSG	NOSORTMSG	x	x
SORTMSG=	SYSOUT	x	x
SORTPGM=	SORT	x	x
SORTSIZE=	MAX	x	x
SORTWKDD=	SASS	x	x
SORTWKNO=	3	x	x
SOURCE	SOURCE	x	x
SOURCE2	SOURCE2	x	x
SPOOL	Batch: NOSPOOL Interactive: SPOOL Noninteractive: NOSPOOL	x	x
STIMER	STIMER	x	
SYMBOLGEN	NOSYMBOLGEN	x	x
SYNCSORT	NOSYNCSORT	x	x
SYSPARM=	"	x	x
SYSIN=	SYSIN	x	x

continued on next page

Table 18.4 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
SYSIN	SYSIN	x	
TAPE=	TAPE	x	x
TAPECLOSE=	REREAD	x	x
TEXT82	TEXT82	x	x
TIME=	MAX	x	x
TLINESIZE=	0	x	x
TPAGESIZE=	0	x	x
TSO	Interactive: TSO Noninteractive: NOTSO	x	x
UNITS=	11 12 13 14 15 16 17 18 19 20	x	x
USER=	WORK	x	x
USERPARM=	"	x	x
VNFERR	VNFERR	x	x
VSAMLOAD	NOVSAMLOAD	x	x
VSAMREAD	NOVSAMREAD	x	x
VSAMUPDATE	NOVSAMUPDATE	x	x
WORK=	WORK	x	
WORKINIT	WORKINIT	x	

VSE Operating System

Table 18.5 Options for the VSE Operating System

Option	Default Value	SAS Invocation	OPTIONS Statement
BATCH/INTERACTIVE	see **Table 18.1**	x	
BAUD=	1200	x	x

continued on next page

Table 18.5 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
BLKSIZE=	0	x	x
BLKSIZE(devicetype)	Optimum for the device	x	x
BUFNO=	2	x	x
BYERR	BYERR	x	x
C60/C48/C96	C96	x	x
CAPS	NOCAPS	x	x
CARDS=	MAX	x	x
CASORT	NOCASORT	x	x
CENTER	Batch: CENTER Interactive: NOCENTER Noninteractive: CENTER	x	x
CHARCODE	NOCHARCODE	x	x
CHKPT	Batch: NOCHKPT Interactive: CHKPT Noninteractive: NOCHKPT	x	x
DAREAD	DAREAD	x	x
DAUPD	NODAAUPD	x	x
DATE	DATE	x	x
DEFAULT	Batch: BATCH Interactive: ICCF Noninteractive: BATCH	x	x
DEVICE=	"	x	x
DMS	NODMS	x	
DQUOTE	DQUOTE	x	x
DSNFERR	DSNFERR	x	
DUMP	NODUMP	x	
DUMPSYS	NOSYSDUMP	x	

continued on next page

Table 18.5 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
ERASE	ERASE	x	x
ERRORABEND	NOERRORABEND	x	x
ERRORS	20	x	x
FILCLR	NOFILCLR	x	x
FILEBLKSIZE(devicetype)	Optimum for the device type	x	x
FILLMEM=	'FFFF'X	x	x
FIRSTOBS=	1	x	x
FMTERR	FMTERR	x	x
FORMCHAR(printdevice)	|----|+|---+=|-/\<>*	x	x
FS	NOFS	x	
FSD=	no default	x	x
FSP	NOFSP	x	
GEN=	5	x	x
GRAPHICS	NOGRAPHICS	x	
ICCF	NOICCF	x	
IMPLMAC	IMPLMAC	x	x
IMS	IMS	x	
INCLUDE	INCLUDE	x	x
INITSTMT=	"	x	x
INVALIDDATA=	.	x	x
LABEL	LABEL	x	x
LAST	_NULL_	x	x
LEAVE	32768	x	x
LINESIZE=	Batch: 132 Interactive: 80 Noninteractive: 132	x	x

continued on next page

Table 18.5 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
LOG=	FT11F001	x	
MACRO	NOMACRO	x	
MACROGEN	NOMACROGEN	x	x
MCOMPILE	NOMCOMPILE	x	x
MEMERR	NOMEMERR	x	x
MEMFILL	NOMEMFILL	x	x
MEMRPT	NOMEMRPT	x	x
MERROR	MERROR	x	x
MISSING=	.	x	x
MLEAVE=	6144	x	
MLOGIC	NOMLOGIC	x	x
MPRINT	NOMPRINT	x	x
MSIZE=	12228	x	
MYMSIZE=	1024	x	
MWORK=	2048	x	
NEWS	NONEWS	x	
NOTES	NOTES	x	x
NUMBER	NUMBER	x	x
OBS=	2147483647	x	x
OFFLINE=	0.0000	x	x
ONLINE=	0.0000	x	x
OPLIST	OPLIST	x	
OVP	Batch: NOOVP Interactive: NOOVP Noninteractive: OVP	x	x

continued on next page

Table 18.5 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
PAGES=	MAX	x	x
PAGESIZE=	60	x	x
PARM=	"	x	x
PARMCARDS=	FT15F001	x	x
PRINTDEVICE=	SYSOUT	x	
PRINTHOVP	NOPRINTHOVP	x	
PRINTINIT	PRINTINIT	x	x
PROBSIG=	0	x	x
REPLACE	REPLACE	x	x
S=	0	x	x
S2=	S	x	x
S370	S370	x	
SASHELP=	H	x	x
SASNEWS=	SASNEWSB	x	x
SEQ=	8	x	x
SERROR	SERROR	x	x
SKIP=	0	x	x
SORTLIST	NOSORTLIST	x	x
SORTPGM=	SORT	x	x
SORTSIZE=	MAX	x	x
SORTWKNO=	1	x	x
SOURCE	SOURCE	x	x

continued on next page

Table 18.5 *Continued*

Option	Default Value	SAS Invocation	OPTIONS Statement
SOURCE2	SOURCE2	x	x
SPOOL	Batch: NOSPOOL Interactive: SPOOL Noninteractive: NOSPOOL	x	x
STIMER	STIMER	x	
SYMBOLGEN	NOSYMBOLGEN	x	x
SYNCSORT	NOSYNCSORT	x	x
SYSPARM=	"	x	x
SYSIN=	SYSIN	x	x
SYSIN	SYSIPT	x	
TAPECLOSE=	REREAD	x	x
TEXT82	TEXT82	x	x
TIME=	MAX	x	x
TPAGESIZE=	0	x	x
UNITS=	11 12 13 14 15 16 17 18 19 20	x	x
USER=	WORK	x	x
USERPARM=	"	x	x
VNFERR	VNFERR	x	x
VSAMLOAD	NOVSAMLOAD	x	x
VSAMREAD	NOVSAMREAD	x	x
VSAMUPDATE	NOVSAMUPDATE	x	x
WORK=	WORK	x	
WORKINIT	WORKINIT	x	

NOTES

1. **CMS and VM/PC**: Options that include an equal sign (=) are specified differently in the OPTIONS statement and in the SAS command (when SAS is invoked). When specified in the OPTIONS statement, the equal sign is used. However, when the option is specified in the SAS command (assuming it can be specified in the SAS command), the equal sign is not specified. For example, the TLINESIZE= option can be specified in an OPTIONS statement or in the SAS command. If specified in an OPTIONS statement, the equal sign must be used:

```
OPTIONS TLINESIZE=120;
```

However, if TLINESIZE is specified in the SAS command, the equal sign is omitted:

```
SAS (TLINESIZE 120
```

2.**OS:** The SAS data library block size value recorded in the data set label (if there is one) is **always** the maximum value permitted for the device. This allows differing block sizes among the SAS data sets in the SAS data library.

VSE: For FBA disks under VSE, the block size is a logical block size.

3. **CMS, OS, VM/PC, and VSE**: The left and right brace substitution characters are not available.

4. **OS**: Checkpointing is useful when using the SAS System interactively under TSO if you want to restart your SAS session after an attention or abend. However, checkpointing increases the cost of a session.

5. **AOS/VS, PRIMOS, and VMS**: When you specify the DMS option, SAS prompts you to enter your teminal device name unless you have specified the device earlier with the FSD= option. You can also invoke the display manager feature with the FSD= option at SAS invocation.

6. **VSE**: Dumping is controlled by the // OPTION [NO | PART]DUMP JCL statement. If you report a problem to SAS Institute, for VSE, use // OPTION PARTDUMP and the DUMPSYS option below.

7. **VSE**: Specify this option when rerunning a failing job in order to obtain a memory dump.

8. **CMS, OS, VM/PC, and VSE**: If you specify NOINCLUDE, SAS execution requires slightly less memory, but you cannot include SAS statements from external files with %INCLUDE. You can still include statements from the internal file of saved statements provided the SPOOL option, below, is in effect.

9. **AOS/VS, PRIMOS, and VMS**: The filename of the disk file is created by appending .LOG to the name of the first SAS source file listed in the SAS command (for batch execution), or the name becomes SASLOG.LOG for a disk file created during interactive execution.

CMS and VM/PC: The filetype of the disk file is SASLOG. The filename of the SASLOG file is determined by the NAME option in the SAS command. If the NAME is not specified, the SASLOG file assumes the filename of the first SAS source file listed in the SAS command. If NAME is not specified and a SAS source file is not specified in the SAS command, the default filename is SAS. This is the default for noninteractive SAS jobs. This option has no effect under display manager. When LDISK is not specified, NOLDISK is in effect.

10. **VSE**: In the event of a console message stating that a VSE system component has insufficient memory, increase the value of LEAVE.

11. **CMS and VM/PC**: When LPRINT is not specified, NOLPRINT is in effect.

12. **CMS and VM/PC**: When LTYPE is not specified, NOLTYPE is in effect.

13. **AOS/VS and VMS**: The filename of the disk file is created by appending .LIS to the name of the first SAS source file listed in the SAS command (for batch

execution), or the name becomes SASLIS.LIS for a disk file created during interactive execution.

CMS and VM/PC: The filetype of the disk file is LISTING. The filename of the LISTING file is determined by the NAME option in the SAS command. If the NAME is not specified, the LISTING file assumes the filename of the first SAS source file listed in the SAS command. If NAME is not specified and a SAS source file is not specified in the SAS command, the default filename is SAS. When PDISK is not specified, NOPDISK is in effect.

PRIMOS: The filename of the disk file is created by appending .LIST to the name of the first SAS source file listed in the SAS command (for batch execution), or the name becomes SASLIS.LIST for a disk file created during interactive execution.

14. **CMS and VM/PC**: This option specifies the CMS sort utility library to be GLOBALed.

OS: The option specifies the data set name that PROC SORT dynamically allocates to fileref SORTLIB. SyncSort does not require a sort library; therefore, when using SyncSort, specify SORTLIB=' '.

15. **VMS**: When SORTPGM=HOST is specified, the sort utility provided by the operating system is used. If the host sort utility is not available or cannot handle the data set, the SAS supplied sort is used. When SORT=BEST is specified, the sort utility best suited for the data is used. This is the default value for this option.

16. **CMS, OS, and VM/PC**: When SORTSIZE=SIZE is specified, the sort executes with 16K less than the total amount of free space in the region or partition. When SORTSIZE=*n* or *n*K is specified, *n* times 1024 bytes of memory is passed to the sort. When SORTSIZE=MAX is specified, the characters MAX are passed to the sort. This causes the system sort to "size" itself. Not all sort programs support this feature. When SORTSIZE=0 is specified, a value of zero is passed to the sort. The sort then uses a value assigned by the installation at the time of sort generation/installation.

VSE: This option has a different meaning and is not normally used. Refer to the *SAS Companion for the VSE Operating System* for more information on sort sizes in the VSE environment.

17. **OS**: The value specified can be a period (.) or a number from 0-6. A period indicates that a default value appropriate for the sort utility in use is to be used. Under OS/MVS, a specification of 0 causes no sort work areas to be allocated and the sort to proceed without them. If you are running a different version of OS, then a specification of 0 permits the sort attempt to proceed even if there are no sort work areas allocated. In either case, when 0 is specified, the system sort utility in use must support "in-core" sorting without sort work areas.

VSE: The value specified can be from 0-9, and it specifies the number of SORTWK files to be used.

18. **AOS/VS**: The resources itemized for each step are I/O blocks, page faults, elapsed time, and CPU time.

OS: The resource itemized for each step is CPU time.

PRIMOS: The resources itemized for each step are I/O time, elapsed time, and CPU time.

VMS: The resources itemized for each step are buffered I/O, direct I/O, page faults, elapsed time, and CPU time.

VSE: The resource itemized for each step is elapsed time.

19. **OS**: If the CLIST option is specified in a TSO environment, the SAS System requests input from the statements in the CLIST following the SAS command, any other stacked input statements, or any other input source identified to TSO, including the terminal.

20. **OS**: If the SYSPARM= option is specified in the JCL EXEC statement OPTIONS= parameter (that is, an EXEC statement that is invoking a cataloged procedure), then four single quotes are required on each side of the option value, as illustrated below:

```
// EXEC SAS,OPTIONS='SYSPARM=''''value'''' more options'
```

VSE: Only two single quotes are required, because the VSE JCL EXEC statement PARM= parameter can only be specified directly. For example,

```
// EXEC SASVSE,PARM='SYSPARM=''value'' more options'
```

21. **AOS/VS, PRIMOS, and VMS**: The only values that can be specified for TAPECLOSE= are LEAVE and REWIND.

CMS: REREAD is equivalent to REWIND, and DISP is ignored.

VSE: DISP is equivalent to REWIND.

22. **OS**: The time measured is execution time.

VSE: The time measured is elapsed time.

23. **AOS/VS, PRIMOS, and VMS**: The file name of the disk file is SAS.SAS.

CMS and VM/PC: The filetype of the file created by the TLOG option is SAS. The filename of the SAS file is determined by the NAME option. If the NAME option is not specified, the filename is also SAS.

24. **OS**: SAS output files use filerefs of the form FT*nn*F001, where *nn* is a unit number, for example, FT11F001.

VSE: Unit numbers are treated as VSE logical unit numbers. Thus, unit 11 means SYS011.

25. **AOS/VS, PRIMOS, and VMS**: the SAS LIBNAME statement must be used to associate the name specified in the USER= option with a directory name.

CMS, OS, VM/PC, and VSE: The operating system control language must define the actual data set name to associate with the name specified in the USER= option.

26. **AOS/VS, PRIMOS, and VMS**: The SAS LIBNAME statement must be used to associate the name specified in the WORK= option with a directory name. The default value, WORK, is automatically associated with your current default directory.

Chapter 19

SAS® Macro Language

Operating systems: All *

* **AOS/VS, PRIMOS, and VMS**: The SAS macro language will be available in a future release of this version of the SAS System.

THE MACRO PROCESSOR

What is it? The macro processor can be used to construct and edit SAS statement text. This processing is done before the text is recognized by the SAS System as a part of DATA or PROC steps. The macro processor is controlled by a special macro language that uses the percent sign (%) and ampersand (&) to denote macro actions. The macro processor simplifies repetitive data entry tasks and provides a way to store and retrieve SAS jobs that must be tailored to changing details. A *macro* is stored text containing SAS code and macro language statements that is referred to by name. The macro facility makes it possible to define macros that accept parameters, accept user input during macro execution, retain macro variables across SAS steps, and conditionally create SAS statements. The SAS macro facility allows you to package long and detailed programs and invoke them with a simple command. You can design a custom language which is translated by your macros to SAS code.

How does it work? When you invoke the SAS System, each SAS statement is first scanned to see if it contains any macro operations. If macro processor actions are present, the macro processor executes them. If the statement does not contain any macro operations, it is passed directly to SAS DATA and PROC step processors. The result of executing macro program statements is to produce SAS code, which is then passed to the SAS System to be executed. **Timing of Macro Facility Actions** later in this chapter discusses the relationship between macro facility execution and SAS execution in more detail.

THE MACRO LANGUAGE

Introductory Example

Suppose, in a long SAS job or session, you type the statements

```
PROC PRINT;
RUN;
```

over and over. The following statements define a macro named P that stores those statements so you can invoke them later without retyping them:

```
%MACRO P;
  PROC PRINT;
  RUN;
  %MEND P;
```

From then on, each time you need to enter those statements you can invoke macro P instead:

```
DATA SAVE.TESTDATA;
   INPUT X;
   CARDS;
data lines
;
%P      ⟵
```

The text that makes up macro P is now substituted for the %P macro call:

```
DATA SAVE.TESTDATA;
   INPUT X;
   CARDS;
data lines
;
PROC PRINT;  ⟵
   RUN;      ⟵
```

Parts of a Macro

The simplest macro definition uses the form:

%MACRO *macroname*;
  macrotext
  %MEND[*macroname*];

The %MACRO statement must begin every macro and must contain a name for the macro.[1] The %MEND statement must close every macro and optionally can specify the macro name to make clear which macro is being closed. See **Specifications** for a discussion of these statements. The macro text includes

1. constant text
2. macro variable references
3. macro function references
4. macro program statements.

Constant text *Constant text* in the macro language is treated as character strings. The text can include SAS variable names, SAS data set names, or all or parts of SAS statements. Constant text is sometimes called *model text* or *model statements* since it is the model, or pattern, for the SAS statements or parts of SAS statements that invoking the macro produces.

Macro variables *Macro variables* (sometimes called *symbolic variables*) belong to the SAS macro language and are different from DATA step variables. A DATA step variable refers to all the data values in a SAS data set for a particular measurement (such as all the values for weight or all the values for X). A macro variable consists of a name and a single value. The value of a macro variable is a character string that may become part of SAS code when processed. The value can change at any time in the SAS job.

The simplest way to define and assign a value to a macro variable is with the macro program statement %LET, as in;

```
%LET name=text;
```

For example, you can write

```
%LET NAME=SAVE.TESTDATA;
```

(See **Specifications** for information on the %LET statement and other macro program statements that create macro variables.)

Rules for naming macro variables are the same as the rules for SAS names: length of one to eight characters; begin with a letter or underscore; letters, numbers, and underscores follow.

The value of a macro variable can range from 0 to 1024 or more bytes in length.[2] A string of length 0 is called a *null string*. A macro variable that has not been assigned a value has a value of null; or you can assign a null value:

```
%LET NAME=;
```

To refer to the value of the variable, specify the name prefixed by an ampersand (&). A macro variable may be referenced inside or outside a macro. The macro processor replaces the macro variable reference with the value of the macro variable (a process called *resolving the macro variable reference* or *symbolic substitution*). Continuing the previous example, you can give the value SAVE.TESTDATA to the macro variable NAME and then reference the variable within a SAS macro:

```
%LET NAME=SAVE.TESTDATA;
%MACRO P;
   PROC PRINT DATA=&NAME;  ←
 RUN;
   %MEND P;
```

When you invoke macro P with the statement %P as before, the value supplied for &NAME is SAVE.TESTDATA:

```
DATA SAVE.TESTDATA;
   INPUT X;
   CARDS;
data lines
;
PROC PRINT DATA=SAVE.TESTDATA;  ←
RUN;
```

You can also reference NAME outside a macro. For example, you can use &NAME as the name of the data set:

```
DATA &NAME;
```

which produces

```
DATA SAVE.TESTDATA;
```

or in a TITLE statement:

```
TITLE "ANALYSIS OF DATA SET &NAME"; ←
```

which becomes

```
TITLE "ANALYSIS OF DATA SET SAVE.TESTDATA"; ←
```

In each case the value of the macro variable NAME is substituted for &NAME in the text. See **Resolving Macro Variable References** later in the chapter for more information, including information on the use of double quotes around the title.

Macro variables are either global or local in scope. A global macro variable can be referenced either inside or outside a macro, and, once created, it exists until the end of the SAS job or session. A local macro variable can be referenced only within the macro in which it was created or a macro nested within that macro, and it exists only while that macro is executing. The area to which a macro variable is available is called its referencing environment. See the descriptions of the %GLOBAL and %LOCAL statements later in this chapter for details on global and local macro variables, and see **Referencing Environments** for an example.

An important kind of local macro variable is the *macro parameter*. You specify macro parameter names and, optionally, default values as part of the %MACRO statement when you define a macro. You supply values for the parameters, or change the default values, when you invoke the macro. The form of a macro with parameters is:

%MACRO *macroname*(*parameter*[,...]);
macrotext containing parameter references
%MEND [*macroname*];

You invoke a macro with parameters as

%*macroname*(*value*[,...])

See the %MACRO statement for more information on macro parameters, including statement-style invocations that use parameters.

SAS also provides a list of automatic macro variables with names beginning with the letters SYS. These automatic macro variables are described later in this chapter.

Macro program statements The macro processor is controlled by *macro program statements*, which are preceded by a percent sign (%). Macro program statements are executed by the macro processor and control what happens inside the macro during macro execution (much as SAS program statements control what happens in a DATA or PROC step). Macro program statements include the following:

%MACRO *name*[(*parameters*)/STMT];
defines a macro

%MEND [*name*];
ends a macro definition

%CMS *command*;
invokes a CMS or CP command

%**comment*;
documents the macro

DO;
begins a DO group

%DO %UNTIL(*expression*);
begins a repetitive DO group

%DO %WHILE(*expression*);
: begins a repetitive DO group

%DO *macrovariable*=*start* %TO *stop* [%BY *increment*];
: begins an iterative DO group

%END;
: ends all types of DO groups

%GLOBAL *macrovariables;*
: declares global variables

%GO TO *label;* or %GOTO *label;*
: skips to label

%IF *expression* %THEN *statement;* [%ELSE *statement;*]
: conditional execution

%INPUT [*macrovariables*];
: requests input from user in conversational environment

%LET *macrovariable*=*value;*
: creates macro variable

%LOCAL *macrovariables;*
: declares local variables

%PUT *text;*
: writes text to SAS log or terminal

%TSO *command;*
: invokes TSO command.

The macro program statements are described later in this chapter under **Specifications**.

Macro expressions Some macro program statements and macro functions (below) use macro expressions—a sequence of macro variable names, constant text, and/or macro function names linked together by operators and, where appropriate, by parentheses. A macro call can also be viewed as an expression. Here are some examples of macro expressions:

```
&X<5
&N*3=&M
&START+&STOP>%LENGTH(&TARGET)
%MACALL(%SUBSTR(&NAME,1,1))=%CHECK(%SUBSTR(&CODE,2,1))
```

Macro expressions can appear in macro variable values, as conditions in statements that test conditions (such as %IF-%THEN and %DO %WHILE statements), and in macro function arguments. When the expression is an arithmetic or logical expression with a numerically valued result, it is evaluated by the macro processor as a number rather than a character string. That is, a numeric expression is evaluated as if the %EVAL function (below) were used.

Macro functions *Macro functions* assist in processing the text used in macros and in macro variable values. Some macro functions correspond to DATA step functions; others perform actions that are needed only within the macro processor. The macro functions include the following:

%BQUOTE
: expresses strings containing special characters.

%EVAL(*expression*)
: evaluates arithmetic and logical expressions.

%INDEX(*argument1,argument2*)
finds the first occurrence of a string.

%LENGTH(*argument*)
finds the length of an argument.

%NRBQUOTE
expresses strings containing special characters.

%NRQUOTE
expresses strings containing special characters.

%NRSTR
expresses strings containing special characters.

%QUOTE
expresses strings containing special characters.

%SCAN(*argument,n*[,*delimiters*])
scans for "words."

%STR(*argument*)
expresses strings containing special characters.

%SUBSTR(*argument,position*[,*length*])
substrings a character string.

%UNQUOTE
restores the meaning of a special character in a string.

%UPCASE(*argument*)
translates lowercase characters to uppercase.

Each macro function is fully described in **Specifications** later in this chapter.

Calling the Macro

Once you have defined a macro, you call (invoke) it with the command

%*macroname*

No semicolon is necessary. When the macro is called, SAS executes the statements generated by the macro of that name. (An alternate form of invocation, called statement-style invocation, is described with the %MACRO statement later in this chapter.)

Keep in mind that macro text may include parts of SAS statements. Thus, when you call the macro, the text supplied by the macro processor must fit in the context of the remaining SAS statements. For example, the macro

```
%MACRO IN(IN1,CUTOFF);
   SET &IN1;
   IF AGE<&CUTOFF;
   RUN;
   %MEND;
```

should only be called after a DATA statement. The following part of a SAS job:

```
DATA CHILD;
   %IN(STORE.DATA,12)
```

generates this complete DATA step for SAS to execute:

```
DATA CHILD;
   SET STORE.DATA;
   IF AGE<12;
RUN;
```

SPECIFICATIONS

Macro program statements

%MACRO *name*[(*parameter*[,...])][/STMT];
%MEND [*name*];
%CMS *command*;
%**comment*;
%DO;
%DO *macrovariable*=*start* %TO *stop* [%BY *increment*];
%DO %UNTIL(*expression*);
%DO %WHILE(*expression*);
%END;
%GLOBAL *macrovariables*;
%GOTO | %GO TO *label*;
%IF *expression* %THEN *statement*;
[%ELSE *statement*;]
%INPUT [*macrovariables*];
%*label*:
%LET *macrovariable*=[*value*];
%LOCAL *macrovariables*;
%PUT *text*;
%TSO *command*;

Macro functions

%BQUOTE(*argument*)
%EVAL(*expression*)
%INDEX(*argument1*,*argument2*)
%LENGTH(*argument*)
%NRBQUOTE(*argument*)
%NRQUOTE(*argument*)
%NRSTR(*argument*)
%QUOTE(*argument*)
%SCAN(*argument*,*n*[,*delimiters*])
%STR(*argument*)
%SUBSTR(*argument*,*position*[,*length*])
%UNQUOTE(*argument*)
%UPCASE(*argument*)

DATA step functions used with the macro language

SYMGET(*argument*)
SYMPUT(*argument1*,*argument2*)

Automatic macro variables

SYSBUFFR
SYSDATE
SYSDAY
SYSDEVIC
SYSDSN
SYSINDEX
SYSENV
SYSPARM
SYSRC

SYSSCP
SYSTIME
SYSVER

Macro Program Statements

%MACRO Statement

The %MACRO statement begins the definition of a macro, assigns the macro a name, and optionally includes a parameter list of macro variables. The %MACRO statement can appear anywhere in the SAS job; however, the macro must be defined before it can be called. Macro definitions may be nested. If two macros are defined with the same name, the second one replaces the first.

The form of the %MACRO statement is

%MACRO *name*[(*parameter*[,...])] [/ **STMT**];

These terms can appear in the %MACRO statement:

name names the macro. *Name* must be a valid SAS name.[3]

parameter specifies a local macro variable for which a value is supplied in the macro call (or in the macro itself). A %MACRO statement can contain any number of macro parameters separated by commas. The macro variables in the parameter list are usually referenced in the macro.

The parameters in the list may be positional or keyword. Define a positional parameter by giving the name of the parameter only in the parameter list. You can define a list of positional parameters in any order, but you must give values for positional parameters in the same order in the macro call as the variables appear in the %MACRO statement. Positional parameters have null values when the macro is defined.

Define a keyword parameter by giving the name of the parameter and an equal sign. You can optionally give a default value after the equal sign; keyword parameters without a value after the equal sign have a default value of null.

If both positional and keyword parameters appear in a macro definition, positional parameters must come first.

STMT specifies that the macro call may be either a name-style invocation or a statement-style invocation. Both types of invocation are discussed below. Macros defined with the STMT option are sometimes called *statement-style macros*.

The IMPLMAC option must be in effect to use statement-style macro calls. Note that if the IMPLMAC option is in effect and you have defined a statement-style macro, SAS checks the first word of every SAS statement to see whether it is a statement-style macro call; when the NOIMPLMAC option is in effect, SAS checks only for the & and % symbols.

Name-style invocations You can use a name-style invocation with any macro. You **must** use a name-style invocation for a macro defined without the STMT

option and for macros defined with the STMT option when the NOIMPLMAC option is in effect.

The form of a name-style invocation is

%*macroname*[([*positionalvalue*, ... *keywordparameter*=*value*, ...])]

The following terms can appear in the invocation:

macroname
is the name of the macro you are invoking.

positionalvalue
specifies a value for a positional parameter. The macro processor matches the first positional parameter value with the first positional parameter in the %MACRO statement, the second with the second, and so on. Thus, the order in which you give positional parameter values in the macro call is important. To omit a positional parameter value at the end of a value list, simply omit it; to omit a positional value that is not the last in the list, mark its place with a comma. Omitting a positional value in a macro invocation causes the parameter to retain its null value.

If an invocation contains both positional and keyword values, the positional values must come first.

keywordparameter=*value*
specifies the name of a keyword parameter followed by a value. Keyword parameters and values can come in any order after the last positional parameter value. Using the parameter name and an equal sign without a value assigns a value of null; omitting the parameter name and value causes the parameter to retain the value assigned in the macro definition. To omit a keyword parameter in the macro call, simply omit it.

To omit the entire list of parameter values in the macro call, omit all the values and the parentheses; all the parameters retain their default values. In addition, in a display manager or interactive session, enter an empty set of parentheses or another word such as a null statement to indicate the absence of the value list.

Here is an example of a name-style invocation:

```
%MACRO BUILD(NEW,IN=INONE);
   DATA &NEW;
      INFILE &IN;
      INPUT A B C;
   PROC PRINT;
      RUN;
   %MEND BUILD;
```

The call

```
%BUILD(APRIL85,IN=INA)
```

yields the output shown in **Output 19.1**.[4]

Output 19.1 Name-Style Invocation with Positional and Keyword Parameter

```
11        + DATA APRIL85;                                                    1-%build
12        + INFILE INA;                                                      1-%build
13        + INPUT A B C;                                                     1-%build
14        + PROC PRINT;                                                      1-%build
15        + RUN;                                                             1-%build
```

Macro BUILD creates a SAS data set from input data in an external file, and then prints the data. The macro variable NEW contains the name of the SAS data set; IN contains the refname of the external file of raw data. The default value for IN is defined to be INONE in the macro definition. However, in the macro call, IN is given the value INA.

In the macro BUILD above, NEW is a positional parameter; its value is given first in the macro call. IN is a keyword parameter.

Macro CREATE has two positional parameters, NEW and OLD:

```
%MACRO CREATE(NEW,OLD);
   DATA &NEW;
      SET &OLD;
      IF PROFIT>0;
   %MEND CREATE;
```

To call CREATE, supply values for NEW and OLD in the same order they were defined:

```
%CREATE(REVENUE,YEAR85)
```

The statements SAS sees are shown in **Output 19.2**.

Output 19.2 Name-Style Invocation with Positional Parameters

```
26           %CREATE(REVENUE,YEAR85)
27        + DATA REVENUE;                                                    1-%create
28        + SET YEAR85;                                                      1-%create
29        + IF PROFIT>0;                                                     1-%create
```

Holding the place of a positional parameter with a comma, as in this example,

```
%CREATE(,YEAR83)
```

produces the statements shown in **Output 19.3**.

Output 19.3 Name-Style Invocation: Holding the Place of a Positional Parameter

```
35        + DATA;                                                            1-%create
36        + SET YEAR83;                                                      1-%create
37        + IF PROFIT>0;                                                     1-%create
```

Macro CHOOSE, below, has two keyword parameters. The macro variable P is given the default value PRINT; T by default has a null value:

```
%MACRO CHOOSE(P=PRINT,T=);
   PROC &P;
   TITLE "&T";
   %MEND CHOOSE;
```

Select the title you want when you call the macro:

```
%CHOOSE(T=PRINTOUT OF DATA)
```

to execute the statements shown in **Output 19.4**.

Output 19.4 Name-Style Invocation with Keyword Parameters

```
47        + PROC PRINT;                                                    1-%choose
48        + TITLE "PRINTOUT OF DATA";                                      1-%choose
```

Both of these calls:

```
%CHOOSE
%CHOOSE()
```

execute the output shown in **Output 19.5**.

Output 19.5 Name-Style Invocation: Omitting a Parameter List

```
53           %CHOOSE
55        + PROC PRINT;                                                    2-%choose
56        + TITLE "";                                                      2-%choose
54           %CHOOSE()
59        + PROC PRINT;                                                    1-%choose
60        + TITLE "";                                                      1-%choose
```

The order for keyword variables in the call is not important. This macro call:

```
%CHOOSE(T=AVERAGE VALUES,P=MEANS)
```

yields **Output 19.6**.

Output 19.6 Name-Style Invocation: Changing the Order of Keyword Parameters

```
65           %CHOOSE(T=AVERAGE VALUES,P=MEANS)
66        + PROC MEANS;                                                    1-%choose
67        + TITLE "AVERAGE VALUES";                                        1-%choose
```

Statement-style invocations Statement-style macro invocations allow you to make macro calls that look like SAS statements. The call begins with a keyword (the macro name), ends with a semicolon, and elements within the call are separated by blanks. You must use a statement-style macro call as a complete statement; the call may not appear within a SAS statement.

The form of a statement-style macro invocation is

macroname [*positionalvalue* ... *keywordparameter*=*value* ... *keywordparameter*];

These terms can appear in a statement-style macro call:

macroname
specifies the name of the macro. Do **not** put a percent sign in front of the name.

positionalvalue
specifies a value for a positional parameter. The macro processor matches the first positional parameter value in the call with the first positional parameter in the macro definition, the second with the second, and so on.

In a statement-style invocation you cannot allow positional parameters to retain their null values by omitting them in the macro call unless the parameters affected are the last parameters in the parameter list. (Commas are treated as characters in the values instead of as delimiters, and a blank cannot hold a place. Thus, omitting a value in the call causes the parameters and values to become mismatched.) To allow a positional parameter to receive a null value, use a %IF statement in the macro definition to conditionally assign the null value:

```
%MACRO TEST(NAME,ID) / STMT;
   %IF &NAME=SKIP %THEN %LET NAME=;
   %PUT &NAME &ID;
   %MEND TEST;
```

The call

```
TEST SKIP 12345;
```

assigns NAME a value of null within the macro, and the %PUT statement writes only the ID number. (To supply a value containing blanks, as in the name MARY WHITE, enclose the value in quotes.)

If a statement-style invocation contains both positional and keyword values, the positional values must come first.

keywordparameter=*value*
specifies the name of a keyword parameter followed by a value. Keyword parameters and values can come in any order after the last positional value. Using the parameter name and an equal sign without a value assigns a value of null; omitting a keyword parameter and value in the macro call causes the parameter to retain the value assigned in the macro definition. To omit a keyword parameter in the macro call, simply omit it.

keywordparameter
specifies that the keyword parameter has a value of 1. Assigning a value to a keyword parameter by specifying only its name is most useful when the default value of the parameter is 0, since 1 has a value of true and 0 has a value of false as in the DATA step. Thus, you can write %IF statements such as

```
%IF &macrovariable %THEN statement;
```

(The %IF statement is discussed later in this chapter.) By assigning a default value of 0 to keyword parameters, you can turn parameters "on" or "off" in a macro call (similar to the way you specify options such as DATE/NODATE in a SAS OPTIONS statement).

Macro SHOW below illustrates referring to a keyword parameter by name only in a statement-style invocation.

Here are examples of statement-style macros and macro invocations. The first example uses a statement-style macro call correctly:

```
%MACRO CHART(TYPE,VAR) / STMT;
   PROC CHART;
      &TYPE &VAR / DISCRETE;
   RUN;
%MEND;

DATA NEW;
   SET OLD;
CHART HBAR SALES;
```

The call

```
CHART HBAR SALES;
```

causes SAS to see the statements shown in **Output 19.7**.

Output 19.7 Statement-Style Invocation with Positional Parameters

```
63        + PROC CHART;                                                    1-%chart
64        + HBAR SALES / DISCRETE;                                         1-%chart
65        + RUN;                                                           1-%chart
```

However, the following SAS program uses a statement-style macro call incorrectly:

```
%MACRO VLIST / STMT;
   AGE HEIGHT WEIGHT
%MEND;

DATA NEW;
   SET OLD;
   KEEP VLIST;
```

Since the name VLIST is not the first word in a statement, the macro processor does not recognize the name as a macro invocation. The text VLIST is passed to SAS without change; therefore, SAS reads VLIST as a variable name and produces an error message if it cannot find that variable.

Macro SHOW is defined with two keyword parameters. Parameter OBS has a default value of 500, and parameter STAY has a default value of 0:

```
%MACRO SHOW(OBS=500,STAY=0) / STMT;
  %IF &STAY %THEN
  %PUT THE VALUE OF OBS IS &OBS AND THE VALUE OF STAY IS &STAY..;
  %ELSE
  %PUT THE VALUE OF OBS IS &OBS AND THE VALUE OF STAY REMAINS &STAY..;
  %MEND SHOW;
```

Referring to STAY by its name only, as in the call

```
SHOW STAY;
```

produces **Output 19.8**.

Output 19.8 Statement-Style Invocation with Keyword Parameter

```
77          SHOW STAY;
THE VALUE OF OBS IS 500 AND THE VALUE OF STAY IS 1.
```

Since the name of the keyword parameter STAY appears in the macro call, STAY receives a value of 1 and the %IF statement is true. Macro parameter OBS, which was not mentioned in the call, retains its default value. (The use of two periods in the macro definition to produce one period in the text is discussed in **Resolving Macro Variable References**.)

Output 19.9 illustrates the effect of other calls.

Output 19.9 Statement-Style Invocation: More Uses of Keyword Parameters

```
84          SHOW OBS;
THE VALUE OF OBS IS 1 AND THE VALUE OF STAY REMAINS 0.
88          SHOW OBS=250;
THE VALUE OF OBS IS 250 AND THE VALUE OF STAY REMAINS 0.
91          SHOW OBS STAY;
THE VALUE OF OBS IS 1 AND THE VALUE OF STAY IS 1.
```

Usage note Do not combine elements of statement-style and name-style macro invocations in the same call, since unexpected results can occur. For example, putting a percent sign in front of a statement-style macro call in the following example:

```
%MACRO NEW(START=0,LIST=) / STMT;
   %IF &START %THEN %LET KEEPVAR=&LIST;
   %ELSE %LET KEEPVAR=MONTH;
   KEEP &KEEPVAR;
%MEND;

DATA THISWEEK;
   SET LASTWEEK;
   %NEW START LIST=WEEK;
```

produces the SAS statements in **Output 19.10**.

In this case, the macro processor reads %NEW as a name-style macro invocation and does not interpret START or LIST as macro parameters. The macro processor produces the KEEP statement indicated when START retains its default value of 0. SAS then reads

```
START LIST=WEEK;
```

as a SAS statement and produces an error message for an invalid statement.

Output 19.10 Combining Name-Style and Statement-Style Invocations

```
11          DATA THISWEEK;
12             SET LASTWEEK;
13             %NEW
14        + KEEP MONTH;                                                    1-%new
13                START LIST=WEEK;
                  ____
                  180
                  180
                  180
16

ERROR 180: STATEMENT IS NOT VALID OR IT IS USED OUT OF PROPER ORDER.

NOTE: SAS STOPPED PROCESSING THIS STEP BECAUSE OF ERRORS.
NOTE: SAS SET OPTION OBS=0 AND WILL CONTINUE TO CHECK STATEMENTS.
      THIS MAY CAUSE NOTE: NO OBSERVATIONS IN DATA SET.
NOTE: DATA SET WORK.THISWEEK HAS 0 OBSERVATIONS AND 1 VARIABLES. 1588 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.34 SECONDS AND 404K.
```

Placing a semicolon after a name-style macro call introduces an extra semicolon into your SAS program. In some cases the additional semicolon becomes a null statement and has no effect; in other cases it causes errors. For example:

```
%MACRO NAME1;
   NAME AGE SEX
%MEND;

%MACRO NAME2;
   HEIGHT WEIGHT
%MEND;
```

If you use these macros in a SAS program, as in

```
PROC PRINT;
   VAR %NAME1; %NAME2;
```

the resulting SAS statements are

```
PROC PRINT;
   VAR NAME AGE SEX; HEIGHT WEIGHT;
```

The variable list for the VAR statement ends prematurely because of the semicolon following the invocation of macro NAME1, and the result of macro NAME2 is an invalid SAS statement.

%MEND Statement

The %MEND statement ends a macro definition. The form of the %MEND statement is

%MEND [*macroname*];

where

macroname optionally names the macro being closed. Repeating the name of the macro for clarity is useful when you create macros nested within other macros.

%CMS Statement

The macro language %CMS statement works as the SAS CMS statement does. It allows you to execute CMS or CP commands immediately and places the return

code for the action into the automatic macro variable SYSRC (described in **Automatic Macro Variables** later in this chapter). Thus, you can test for the successful completion of the command. You can use the %CMS statement inside or outside a macro.

The form of the %CMS statement is

%CMS *command*;

where

command may be any CMS or CP command or any sequence of macro operations that generates a CMS or CP command.

Omitting the command puts you into CMS subset mode. See the CMS statement for details.

For example:

```
%CMS FILEDEF IN DISK MYFILE DATA NEW;
%PUT &SYSRC;
```

If you are not running under CMS, the %CMS command is ignored.

%*comment Statement

Use the macro comment statement when you want to place comments in the macro. No text is produced from the comments as it is from SAS comment statements, which the macro language treats as constant text. Macro comment statements can appear only inside a macro.

The form of the %*comment statement is

%*comment*;

Note: macro variable references are not resolved if they appear in a SAS comment statement.

For example, the following macro produces code to check for errors in the data:

```
%MACRO VERIFY(IN);
   %*                                        ;
   %*****MACRO FOR QUICK DATA CHECKING****;
   %*                                        ;
   DATA CHECK;
      INFILE &IN;
      INPUT X Y Z;
      IF X<0 OR Y<0 OR Z<0 THEN LIST;
      *CHECKS FIRST THREE VARIABLES IN FILE &IN;
      RUN;
   %MEND VERIFY;
```

When you call macro VERIFY with the statement

```
OPTIONS MACROGEN SYMBOLGEN;
%VERIFY(INA)
```

SAS sees the statements in **Output 19.11.**

Output 19.11 Macro Comment Statement and Macro Variable Reference in SAS Comment Statement

```
14          %VERIFY(INA)
17        +DATA CHECK;                                                    1-%verify
19        +INFILE &IN                                                     1-%verify
20        + INA                                                           2-&SYMBOL
19        +       ;                                                       1-%verify
22        +INPUT X Y Z;                                                   1-%verify
24        +IF X<0 OR Y<0 OR Z<0 THEN LIST;                                1-%verify
26        +*CHECKS FIRST THREE VARIABLES IN FILE &IN;                     1-%verify
28        +RUN;                                                           1-%verify
```

In the SAS comment statement, the macro variable reference &IN is unresolved.

%DO Statement

The %DO statement works in the macro language as the simple DO statement works in the DATA step. Text and program statements following the %DO are processed until a matching %END appears.

The form of the %DO statement is

%DO;
text and macro program statements
%END;

For example, the macro

```
%MACRO DO1(STATE);
   %IF %UPCASE(&STATE)=NC %THEN %DO;
      TITLE "STATE OF NORTH CAROLINA";
      FOOTNOTE "1985 DATA";
      %END;
   %ELSE %DO;
      TITLE;
      FOOTNOTE;
      %END;
   %MEND DO1;
```

uses %DO groups to issue one of two sets of TITLE and FOOTNOTE statements. (The %UPCASE function, which converts lowercase characters to uppercase, is described in **Macro Functions**.) Some possible calls and the statements they produce are shown in **Output 19.12**.

Output 19.12 %DO Statement

```
144         %DO1(NC)
145       + TITLE "STATE OF NORTH CAROLINA";                              1-%do1
146       + FOOTNOTE "1985 DATA";                                         1-%do1
148         %DO1(nc)
149       + TITLE "STATE OF NORTH CAROLINA";                              1-%do1
150       + FOOTNOTE "1985 DATA";                                         1-%do1
152         %DO1(Nc)
153       + TITLE "STATE OF NORTH CAROLINA";                              1-%do1
154       + FOOTNOTE "1985 DATA";                                         1-%do1
156         %DO1(nC)
157       + TITLE "STATE OF NORTH CAROLINA";                              1-%do1
158       + FOOTNOTE "1985 DATA";                                         1-%do1
160
```

Iterative %DO Statement

The iterative %DO statement works as the iterative DO does in the DATA step except that in the iterative %DO statement:

- the index variable is a macro variable
- you cannot replace *start* and *stop* with a list of values as you can in the DATA step
- the *start, stop,* and *increment* values may be any valid macro expressions.

The form of the iterative %DO statement is

%DO *macrovariable*=*start* **%TO** *stop* [**%BY** *increment*];
text and macro program statements
%END;

This example shows the use of the iterative %DO statement:

```
%MACRO VARLIST(NUMBER);
   VAR
      %DO X=2 %TO &NUMBER %BY 2;
         VAR&X
         %END;
      ;
   %MEND HEADING;
```

The call

```
%VARLIST(6)
```

produces the SAS statements in **Output 19.13**.

Output 19.13 Iterative %DO Statement

```
43          %VARLIST(6)
44        + VAR VAR2 VAR4 VAR6 ;                                                    1-%varlist
```

The following example uses macro INTERVAL to provide an increment for the iterative %DO loop in macro VARLIS2:

```
%MACRO INTERVAL;
   %*MACRO INTERVAL SELECTS AN INTERVAL FOR MACRO VARLIST2;
   %IF &NUMBER<=5 %THEN 1;
   %ELSE %IF &NUMBER<=10 %THEN 2;
   %ELSE 5;
   %MEND INTERVAL;
%MACRO VARLIS2(NUMBER);
   VAR
   %DO X=1 %TO &NUMBER %BY %INTERVAL;
       VAR&X
       %END;
       ;
   %MEND VARLIS2;
```

Suppose you want to print a subset of the 50 variables (named VAR1-VAR50) in data set BIG. Use these statements:

```
PROC PRINT DATA=BIG;
   %VARLIS2(50)
```

The statements shown in **Output 19.14** are produced.

Output 19.14 Iterative %DO Statement Containing Macro Call

```
52          PROC PRINT DATA=BIG;
53             %VARLIST(50)
54        + VAR VAR1 VAR6 VAR11 VAR16 VAR21 VAR26 VAR31 VAR36 VAR41 VAR46 ;          1-%varlis2
```

Every fifth variable in BIG appears in the list of variables. (The %IF-%THEN statements are described below.)

If the macro variable used as the index has the same name as another macro variable available to the local environment, the %DO statement changes the value of the previously defined macro variable. An unnoticed conflict in macro variable names can thus lead to unexpected results. This example:

```
OPTIONS MPRINT SYMBOLGEN;
%LET I=INDUSTRIAL PRODUCTION;
TITLE "REPORT ON &I AT VARIOUS TIMES";
%MACRO CREATE;
   %DO I=1 %TO 3;
      DATA DATA&I;
         INFILE IN&I;
         INPUT PRODUCT COST DATE;
      RUN;
      %END;
   %MEND CREATE;
%CREATE
TITLE "REPORT ON LATEST &I";
PROC PRINT DATA=DATA3;
```

produces the SAS statements in **Output 19.15**.

Output 19.15 Value of Macro Variable Used as Index Variable

```
74          %CREATE
75        + DATA DATA1;                                                          1-%create
76        + INFILE IN1;                                                          1-%create
77        + INPUT PRODUCT COST DATE;                                             1-%create
78        + RUN;                                                                 1-%create


NOTE: INFILE IN1 IS:
      DSNAME=XXXXXX.XXXXX.XXXX,
      UNIT=DISK,VOL=SER=XXXXXX,DISP=SHR,
      DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB)

NOTE: 1 LINE WAS READ FROM INFILE IN1.
NOTE: DATA SET WORK.DATA1 HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 1.10 SECONDS AND 360K.
```

(continued on next page)

(continued from previous page)

```
79         + DATA DATA2;                                                    1-%create
80         + INFILE IN2;                                                    1-%create
81         + INPUT PRODUCT COST DATE;                                       1-%create
82         + RUN;                                                           1-%create


NOTE: INFILE IN2 IS:

      DSNAME=XXXXXX.XXXXX.XXXX,
      UNIT=DISK,VOL=SER=XXXXXX,DISP=SHR,
      DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB)

NOTE: 1 LINE WAS READ FROM INFILE IN2.
NOTE: DATA SET WORK.DATA2 HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 1.15 SECONDS AND 368K.
      THE COMPILE   PHASE USED 0.94 SECONDS.
      THE EXECUTION PHASE USED 0.21 SECONDS.

83         + DATA DATA3;                                                    1-%create
84         + INFILE IN3;                                                    1-%create
85         + INPUT PRODUCT COST DATE;                                       1-%create
86         + RUN;                                                           1-%create


NOTE: INFILE IN3 IS:
      DSNAME=XXXXXX.XXXXX.XXXX,
      UNIT=DISK,VOL=SER=XXXXXX,DISP=SHR,
      DCB=(BLKSIZE=3120,LRECL=80,RECFM=FB)

NOTE: 1 LINE WAS READ FROM INFILE IN3.
NOTE: DATA SET WORK.DATA3 HAS 1 OBSERVATIONS AND 3 VARIABLES. 680 OBS/TRK.
NOTE: THE DATA STATEMENT USED 1.18 SECONDS AND 368K.

88            TITLE "REPORT ON LATEST &I";
89
89            PROC PRINT DATA=DATA3;
94          RUN;
NOTE: THE PROCEDURE PRINT USED 1.15 SECONDS AND 468K AND PRINTED PAGE 1.
```

```
                                REPORT ON LATEST 4                                1

                          OBS    PRODUCT    COST    DATE

                           1       8855      1500    1985
```

In the second TITLE statement the current value of the macro variable I is substituted. Thus the title "REPORT ON LATEST 4," **not** "REPORT ON LATEST INDUSTRIAL PRODUCTION," is printed. (See the DO statement in "Statements Used in the DATA Step" for a discussion of the value of an index variable after the last execution of a loop.)

To avoid this result either change the name of the index variable or use a %LOCAL statement to make a local macro variable I for CREATE. (See a description of the %LOCAL statement, below.)

%DO %UNTIL and %DO %WHILE Statements

The %DO %UNTIL and %DO %WHILE statements work as the DO WHILE and DO UNTIL statements do in SAS processing.

The form of the %DO %UNTIL and %DO %WHILE statements is

%DO %UNTIL(*expression*)**;**
text and macro program statements
%END;

%DO %WHILE(*expression*)**;**
text and macro program statements
%END;

Expression may be any valid macro expression. The macro processor evaluates the expression, substituting macro variable values or the result of a macro call. For example

```
%DO %WHILE(&A+&B<&C);
%DO %WHILE(%BUILD);
%DO %UNTIL(&HOLD=NO);
```

are acceptable statements.

%END Statement

The %END statement ends a %DO group. The form of the %END statement is

%END;

A %END statement must end every %DO group in the job or session. For example,

```
%MACRO TEST(FINISH);
   %LET I=1;
   %DO %WHILE (&I<&FINISH);
       %PUT THE VALUE OF I IS &I;
       %LET I=%EVAL(&I+1);
       %END;
   %MEND TEST;
```

See the %DO statement description earlier in this section for more examples using %END; see **Macro Functions** for a description of the %EVAL function.

%GLOBAL Statement

The %GLOBAL statement is used to create global macro variables, that is, macro variables that are available to all referencing environments of a job (except where replaced by local macro variables of the same names). See **Referencing Environments** for a discussion of referencing environments. The %GLOBAL statement can appear anywhere in a SAS job; it is an executable statement.

The form of the %GLOBAL statement is

%GLOBAL *macrovariables*;

where

macrovariables are the names of global macro variables. For example, this %GLOBAL statement creates three global macro variables:

```
%GLOBAL DATAVAR KEEP1 KEEP2;
```

Only macro variables named in a %GLOBAL statement or those defined outside all macros are global. Once you have defined a macro variable as local, you cannot use a %GLOBAL statement in the same environment to make it global. Thus, if you want to create a global macro variable within a macro, specify it in a %GLOBAL statement before you use it in any other macro program statements.

Macro variables created with a %GLOBAL statement have null values until you assign them other values. If you have already created a global macro variable and assigned it another value, including that variable in a %GLOBAL statement does not remove the value. For example, you may want to keep track of global macro variables in a job by recording them all in a %GLOBAL statement at some point.

In this example macro VARIABL produces a global macro variable whose value represents one of two lists of variables:

```
OPTIONS MPRINT SYMBOLGEN;
%MACRO VARIABL(TYPE);
   %GLOBAL LIST;
   %IF &TYPE=1 %THEN %LET LIST=PASS FAIL;
   %ELSE %IF &TYPE=2 %THEN %LET LIST=INSPECTR STATION SAMPLE;
   %MEND VARIABL;
```

In the listing in **Output 19.16** each PROC step uses a different value of the macro variable LIST as a variable list for the VAR statement.

Output 19.16 %Global Statement

```
105        %VARIABL(1)
109        PROC PRINT;
110           VAR &LIST
111        + PASS FAIL                                                    1-&SYMBOL
110                    ;
112        RUN;
113        %VARIABL(2)
116        PROC PRINT;
117           VAR &LIST
118        + INSPECTR STATION SAMPLE                                      1-&SYMBOL
117                    ;
119        RUN;
```

If the %GLOBAL statement were not included in macro VARIABL, the value of LIST assigned within the macro would not be available to the rest of the job. An error message reporting that &LIST is not a valid name for a SAS variable would be printed.

%GOTO or %GO TO Statement

The %GOTO or %GO TO statement causes the macro processor to branch to the label specified in the %GOTO or %GO TO statement. The statements work the same way as the GOTO and GO TO statements in the DATA step.

The form of the %GOTO statement is

%GOTO | **%GO TO** *label*;

where

label is the label of a statement to which you want execution to branch. See the **%label: Statement** later in this chapter for information on statement labels in the macro language. The %GOTO statement and the statement label must be in the same referencing environment.

You can use a macro variable reference or a macro call to produce the label in a %GOTO statement during macro execution. A %GOTO statement containing a macro variable reference or a macro call is sometimes called a *computed %GOTO*. A computed %GOTO statement creates a local referencing environment; see **Referencing Environments** for details.

Here are examples of valid %GOTO statements:

```
%GOTO XXX;
%GO TO NAME1;
```

```
%GOTO &WHERE;
%GO TO %FIND;
```

Note: do **not** put a % sign in front of the label in a %GOTO statement unless you intend to call a macro of that name.

%IF-%THEN/%ELSE Statements

The %IF-%THEN and %ELSE statements work like the corresponding statements IF-THEN and ELSE in the SAS DATA step. However, you cannot use %IF without %THEN; thus, you cannot use %IF to create subsets as you do in the DATA step.

The form of the %IF-%THEN and %ELSE statements is

%IF *expression* **%THEN** *statement;*
%ELSE *statement;*

where

expression is any valid macro expression. The macro processor evaluates the expression, substituting macro variable values or the result of a macro call. If the expression is true (has a nonzero value) the %THEN is processed. If the expression is false (zero) the %ELSE, if one is present, is processed.

statement is any macro program statement, constant text, expression, or macro call. If *statement* includes constant text containing semicolons, enclose the semicolons in the %STR function, described in **Macro Functions**.

For example:

```
%MACRO FISCAL;
   %IF "&SYSDATE"="15APR85" %THEN %DO;
      PROC MEANS DATA=TOTAL;
      PROC CHART DATA=TOTAL;
         HBAR REVENUE / TYPE=MEAN;
      %END;
   %ELSE %DO;
      PROC MEANS DATA=MONTHLY;
      PROC PRINT DATA=MONTHLY;
      %END;
   %MEND;
```

If the value of the automatic macro variable SYSDATE is 15APR85, the expression is true, and macro FISCAL processes the statements following the %THEN. If the value of SYSDATE is not 15APR85, the statements following the %ELSE are generated.

Enclosing both &SYSDATE and the comparison string in double quotes avoids problems that can occasionally arise when a string beginning with a number 0-9 followed by a letter A-F is interpreted as a hexadecimal number instead of as a character string.

%INPUT Statement

The %INPUT statement allows you to supply values to macro variables during macro execution; it permits the construction of conversational macros. When the macro processor reaches a %INPUT statement during macro execution, it waits for you to enter values at the terminal for the macro variables named in the state-

ment. The %INPUT statement can be used inside or outside a macro. (Note: the %INPUT statement is not currently supported in batch mode.)

The form of the %INPUT statement is

%INPUT [*name* ...];

where

name may be a macro variable name, or a macro variable reference or macro call that produces a macro variable name. The %INPUT statement can contain any number of names separated by blanks.

The value for each macro variable must be a single item, and values for macro variables must be separated by blanks. If the value contains blanks, enclose the value in quotes, as in

```
'MARY GREEN'
```

The macro processor matches macro variable values with macro variable names in order. Therefore, if you supply more than one item as the value of a macro variable, the second item becomes the value of the next macro variable; the macro variables and values become mismatched. After all values have been matched with macro variable names, excess text becomes the value of the automatic macro variable SYSBUFFR (see **Automatic Macro Variables** later in this chapter). If a %INPUT statement does not contain any macro variable names, all text entered is assigned to SYSBUFFR.

The following macro requests you to input values for two macro variables. Excess text is assigned to SYSBUFFR. Three calls are shown:

```
%MACRO PLACES;
   %PUT ENTER VALUES FOR CITY AND STATE.;
   %INPUT CITY STATE;
   %PUT *&CITY* **&STATE** ***&SYSBUFFR***;
   %MEND PLACES;
```

See **Output 19.17**.

Output 19.17 %INPUT Statement and Automatic Macro Variable SYSBUFFR

```
  6? %places
 ENTER VALUES FOR CITY AND STATE.
Reno Nevada
 *Reno* **Nevada** ******
  7? %places
 ENTER VALUES FOR CITY AND STATE.
San Francisco California
 *San* **Francisco** *** California***
  8? %places
 ENTER VALUES FOR CITY AND STATE.
'San Francisco' California
 *'San Francisco'* **California** ******
```

In the first call RENO is the value of CITY, NEVADA is the value of STATE, and SYSBUFFR has a null value (0 characters).

In the second call SAN is the value of CITY, FRANCISCO is the value of STATE, and CALIFORNIA (beginning with the blank following FRANCISCO) is the value of SYSBUFFR.

In the third call 'SAN FRANCISCO' is the value of CITY, CALIFORNIA is the value of STATE, and SYSBUFFR has a null value. Note that the quotes around 'SAN FRANCISCO' are part of the value of CITY; to remove the quotes, use the %SUBSTR function, as in

```
%LET CITY2=%SUBSTR(&CITY,2,%LENGTH(&CITY)-2);
```

The value of CITY2 is

```
SAN FRANCISCO
```

If you need to input only one macro variable value containing blanks, you can allow the entire value to be assigned to SYSBUFFR and use the value of SYSBUFFR as the value of a new macro variable. In this example:

```
%MACRO CITIES;
   %PUT ENTER A VALUE FOR CITY;
   %INPUT;
   %LET CITY=&SYSBUFFR;
   %PUT *&CITY*;
   %MEND CITIES;
```

calling macro CITIES produces the results shown in **Output 19.18**.

Output 19.18 Assigning Input Directly to Macro Variable SYSBUFFR

```
  15? %cities
 ENTER A VALUE FOR CITY
Reno
 *Reno*
  16? %cities
 ENTER A VALUE FOR CITY
San Francisco
 *San Francisco*
```

%label: Statement

Statement labels in the macro language are the same as statement labels in the DATA step, and the rules for naming labels in the macro language are also the same.

The form of macro statement labels is

%*label*: *statement*;

where

label is any valid SAS name.

statement is a macro program statement, macro variable reference, macro call, or piece of constant text. Here are some examples of valid statement labels in the macro language:

```
%ONE: %LET BOOK=ELEMENTARY;
%FINAL: DATA _NULL_;
```

When you label a statement in the macro language, put a % sign in front of the label. When you use the label in a %GOTO statement, do not use a % sign. A % sign in front of a label name in a %GOTO statement tells the macro processor to call a macro of that name. If the macro processor cannot find the macro called, an error message is issued.

This example illustrates the use of labels in the macro language. Some possible calls and the statements they produce are listed in **Output 19.19**.

```
%GLOBAL SWITCH;
%MACRO INFO(LONG,HIST);
   %IF &SWITCH=1 %THEN %GOTO SKIP;

   %MACRO CHOOSE;
      %*THIS MACRO CHOOSES A LABEL FOR MACRO INFO;
      %IF &HIST=HIST %THEN LABEL1;
      %ELSE LABEL2;
      %MEND CHOOSE;

   %LET SWITCH=1;
%SKIP:
   %IF &LONG=LONG %THEN %GOTO %CHOOSE;
   %ELSE %GOTO LABEL3;
   %LABEL1: PROC CONTENTS HISTORY;
   %LABEL2: PROC FREQ;
               TABLES _NUMERIC_;
   %GOTO LABEL4;
   %LABEL3: PROC PRINT DATA=_LAST_(OBS=10);
   %LABEL4: RUN;
   %MEND INFO;
```

Output 19.19 %label Statement

```
240          %INFO(LONG,HIST)
241       + PROC CONTENTS HISTORY                                    1-%info
243       + PROC FREQ;                                               1-%info
244       + TABLES (_NUMERIC_);                                      1-%info
245       + RUN;                                                     1-%info
247          %INFO(LONG)
248       + PROC FREQ;                                               1-%info
249       + TABLES (_NUMERIC_);                                      1-%info
250       + RUN;                                                     1-%info
252          %INFO(SHORT)
253       + PROC PRINT DATA=                                         1-%info
254       +_LAST_(OBS=10);                                           1-%info
255       + RUN;                                                     1-%info
257          %INFO
259       + PROC PRINT DATA=                                         2-%info
260       +_LAST_(OBS=10);                                           2-%info
261       + RUN;                                                     2-%info
258          %INFO(,HIST)
264       + PROC PRINT DATA=                                         1-%info
265       +_LAST_(OBS=10);                                           1-%info
266       + RUN;                                                     1-%info
```

Macro INFO illustrates several features.

1. The macro variable SWITCH causes the inner macro CHOOSE to be compiled only once. Since SWITCH is global, the value of 1 assigned to it in the first execution of macro INFO is preserved until the end of the job or session unless you change it. See the %GLOBAL statement earlier in this chapter for more information.
2. The macro call %CHOOSE appears in the %GOTO statement and generates the appropriate label based on the parameter values specified in the call of macro INFO.

3. In the first call, the values of macro variables LONG and HIST are the same characters as the variable names.
4. The label %SKIP: labels a macro program statement, while labels %LABEL1-%LABEL4 label constant text.
5. The special SAS data set name _LAST_ (defined as the most recent SAS data set created) allows you to use the data set option OBS= without knowing the name of the data set.

%LET Statement

Use the %LET statement either inside or outside a macro to create a macro variable and assign it a value or to change the value of an existing macro variable. A %LET statement can define only one macro variable.

The form of the %LET statement is

%LET *macrovariable*=[*value*];

where

macrovariable is the name of a new or existing macro variable.

value is the value of the macro variable. Omitting *value* produces a value of null (0 characters).

This example uses %LET to create a macro variable named STATE with a value of NC:

```
%LET STATE=NC;
```

Macro LISTING references STATE:

```
%MACRO LISTING;
   PROC PRINT DATA=&STATE;
   TITLE "LISTING OF &STATE RESIDENTS";
   %MEND;
```

You can supply values with a %LET statement before calling the macro:

```
%LET STATE=NC;
%LISTING
%LET STATE=SC;
%LISTING
```

%LOCAL Statement

The %LOCAL statement specifies macro variables that are to be local to the macro. The statement is useful for ensuring that values of macro variables that may have been specified earlier are not inadvertently used in the current macro. The %LOCAL statement may be specified only inside the definition of a macro; the statement is executable.

The form of the %LOCAL statement is

%LOCAL *macrovariables*;

where

macrovariables are the names of local macro variables. For example, this %LOCAL statement specifies two macro variables:

```
%LOCAL TOTAL1 TOTAL2;
```

Variables created with a %LOCAL statement have null values until you assign them another value. However, if a previously defined local macro variable is

included in a %LOCAL statement, the macro variable retains its previously assigned value.

In the following example, macro PRINTIT defines a local macro variable named COUNT:

```
%MACRO PRINTIT(STOP);
   %LOCAL COUNT;
   %LET COUNT=1;
   %DO %WHILE (&COUNT<&STOP AND &STOP<=9999);
      PROC PRINT DATA=DATA&COUNT;
      %LET COUNT=%EVAL(&COUNT+1);
      %END;
   %MEND PRINTIT;
```

If COUNT is also defined as a macro variable outside macro PRINTIT, the previously defined value does not influence the value of COUNT in macro PRINTIT. The %EVAL function, which increments COUNT's value, is described in **Macro Functions** later in this chapter. A macro call and the statements produced are shown in **Output 19.20**.

Output 19.20 %LOCAL Statement

```
10         OPTIONS MACROGEN SYMBOLGEN;
11         %PRINTIT(5)
14        +PROC PRINT DATA=DATA&COUNT                                    1-%printit
15        +DATA1                                                         2-&SYMBOL
14        +                        ;                                     1-%printit
17        +PROC PRINT DATA=DATA&COUNT                                    1-%printit
18        +DATA2                                                         2-&SYMBOL
17        +                        ;                                     1-%printit
20        +PROC PRINT DATA=DATA&COUNT                                    1-%printit
21        +DATA3                                                         2-&SYMBOL
20        +                        ;                                     1-%printit
23        +PROC PRINT DATA=DATA&COUNT                                    1-%printit
24        +DATA4                                                         2-&SYMBOL
23        +                          ;                                   1-%printit
```

For details on how the macro processor resolves strings like DATA&COUNT and others containing macro variables, see **Resolving Macro Variable References** later in this chapter.

A %LOCAL statement can be executed conditionally. For example:

```
%LET TIME=YEAR;
TITLE "CUMULATIVE REPORT FOR &TIME";
%MACRO SUBSETS(MONTH);
   %*REMEMBER TIME IS A GLOBAL MACRO VARIABLE BECAUSE IT WAS CREATED
      OUTSIDE THE MACRO;
   %IF %UPCASE(&MONTH)¬=DECEMBER %THEN %DO;
      %LOCAL TIME;
      %*NOW YOU HAVE CREATED A LOCAL MACRO VARIABLE NAMED TIME;
      %LET TIME=&MONTH;
      TITLE2 "COVERING JANUARY--&TIME";
      %END;
   %ELSE %STR(TITLE2 "REPORT FOR &TIME";);
   RUN;
   %MEND SUBSETS;
```

When the macro SUBSETS is called, if the uppercase equivalent of the value supplied for MONTH is anything other than DECEMBER, a local variable TIME

is created. The MONTH value given is assigned to the local macro variable TIME and used in the TITLE2 statement. If the uppercase equivalent of the value is DECEMBER, the value YEAR from the global macro variable TIME is used in the TITLE2 statement. (The %UPCASE function translates a lowercase value into uppercase, and the %STR function prevents the semicolon at the end of the TITLE2 statement from closing the %ELSE statement. See **Macro Functions** for details.) Thus, the calls below produce the statements indicated in **Output 19.21**.

Output 19.21 Conditional Execution of a %LOCAL Statement

```
52          %SUBSETS(DECEMBER)
53         +TITLE2 "REPORT FOR YEAR";                                    1-%subsets
54         + RUN;                                                        1-%subsets
56          %SUBSETS(OCTOBER)
57         + TITLE2 "COVERING JANUARY--OCTOBER";                         1-%subsets
58         + RUN;                                                        1-%subsets
```

%PUT Statement

The %PUT statement writes single lines of text to the SAS log if you are working in batch mode and to the terminal if you are working interactively. It can be used inside or outside a macro. Note: the %PUT statement in the macro facility replaces the %PUT statement documented in "SAS Statements Used Anywhere." However, if you disable the macro facility by specifying the NOMACRO option, the %PUT statement documented in "SAS Statements Used Anywhere" is available.

The form of the %PUT statement is

%PUT *text;*

where

text is any text. *Text* is printed exactly as it appears. If *text* is longer than the current line size, the additional characters up to a length of 132 are printed on the next line. *Text* longer than 132 characters is truncated.

To quote text written by a %PUT statement use the macro language quoting functions (see **Macro Quoting** for details). This macro illustrates quoting in several %PUT statements:

```
%MACRO TESTPUT;
   %PUT %STR(USE A SEMICOLON(;) TO END A SAS STATEMENT);
   %PUT %STR( );
   %PUT %STR(ENTER THE STUDENT%'S ADDRESS);
   %MEND TESTPUT;
```

See **Output 19.22**.

Output 19.22 %PUT Statement

```
295
296          %TESTPUT
USE A SEMICOLON(;) TO END A SAS STATEMENT

ENTER THE STUDENT'S ADDRESS
```

%TSO Statement

The %TSO statement works as the SAS TSO statement does and can be used inside or outside a macro. The %TSO statement allows you to execute TSO commands immediately and places the return code for the action into the automatic macro variable SYSRC. Thus, you can test for the successful completion of the command. You can issue any TSO command with the %TSO statement except the TEST, LOGON, or LOGOFF commands, or any command that must be given control in an authorized state.

The form of the %TSO statement is

%TSO [*command*]**;**

where

command is a TSO command or any sequence of macro operations that produces a TSO command.

Omitting the command places you into TSO submode. See the TSO statement in "SAS Statements Used Anywhere" for details.

For example:

```
%MACRO SASDATA;
   %TSO ALLOC FI(IN) DA('USERID.MY.SASDATA') OLD;
   %IF &SYSRC¬=0 %THEN
         %TSO ALLOC FI(IN) DA('USERID.MY.SASDATA') SHR;
   %MEND SASDATA;
```

If you are not running under OS, the TSO statement is ignored.

Macro Functions

Macro functions process one or more character strings or macro expressions, called *arguments*, to produce a new character string, the *result* or the *returned value*. Macro functions can be used within a macro or within a macro program statement used outside a macro (such as %LET or %PUT). Macro functions can be divided into three categories: character functions, the evaluation function, and quoting functions.

Macro character functions Macro character functions have names corresponding to DATA step character functions (such as %SUBSTR and SUBSTR); the logic and purpose of the two are similar. However, the macro function works during macro execution (or the execution of a macro program statement such as %LET outside a macro), that is, before SAS executes the DATA step. The DATA step function works during the execution of the DATA step. A DATA step function such as SUBSTR appears as constant text to the macro processor.

The macro character functions are as follows:

%INDEX finds the first occurrence of a string.

%LENGTH finds the length of an argument.

%SCAN	scans an argument for "words".
%SUBSTR	substrings a character string.
%UPCASE	translates lowercase characters to uppercase.

Macro evaluation function The evaluation function is %EVAL. Since the macro processor is a character-handling facility, "numbers" in macro expressions are treated as character strings. The %EVAL function assigns numeric properties to integers in macro expressions and performs calculations using integer arithmetic. (Calculations on fractions are not allowed; the result of an integer calculation such as division that results in a fraction is truncated to an integer.) The %EVAL function also evaluates logical expressions such as &A=&B.

Macro quoting functions The quoting functions in the macro language perform the activity equivalent to enclosing a portion of a SAS statement in single or double quotes. That is, the macro facility treats the expression within the function as a unit and does not perform any evaluations on the expression. (The single quote (') and double quote (") characters are treated as text within the macro facility.) There are six quoting functions within the macro facility, divided into three pairs, and one function that reverses the effect of the other six. The quoting functions and their activity can be outlined as follows:

Table 19.1 Relationships Among Macro Quoting Functions

Time when the function takes effect	Removes the meaning from special characters except % and &	Removes the meaning of these characters also
Macro compilation	%STR( )	%NRSTR( )
Macro execution	%QUOTE( ) %BQUOTE( )	%NRQUOTE( ) %NRBQUOTE( )

The %UNQUOTE function restores the meaning that was removed from a special character by one of the functions above.

See **Quoting in the Macro Facility** under **Special Topics** for more information on quoting in the macro facility.

All functions are described individually in alphabetical order below.

%BQUOTE: removes meaning from unanticipated special characters (except %, &, and mnemonic operators) during macro execution

%BQUOTE(*argument*)

The %BQUOTE (**B**lind **Quote**) function is similar to the %QUOTE function. It removes the meaning from special characters during macro execution; however, it does not remove the meaning from the ampersand (&), the percent sign (%), or strings that are also operators (such as AND and OR). The %BQUOTE function is useful for manipulating text that may contain unmatched quotes or parentheses that you cannot anticipate. For example, suppose a macro contains these statements:

```
%PUT ENTER THE LOCATION OF THE MEETING.;
%INPUT;
```

A response such as

```
DOUG'S OFFICE
```

contains an unmatched single quote. You can have the macro facility process this text by enclosing the text in the %BQUOTE function, as in

```
%LET PLACE=%BQUOTE(&SYSBUFFR);
```

The unmatched single quote in the value of PLACE does not cause errors.

%EVAL: evaluates arithmetic and logical expressions

%EVAL(*expression*)

The %EVAL function evaluates arithmetic and logical expressions in the macro language. The function may be used both inside and outside of macros. Note that all statements in the macro language which evaluate expressions contain an implied %EVAL that evaluates the condition and returns a value of 1 if the condition is true and 0 if it is false.

Suppose for example that the statement

```
%LET X=2;
```

appears in a SAS job. Then the statement

```
%LET Y=%EVAL(&X+1);
```

assigns Y a value of 3. The %EVAL function allows the value of macro variable X to have numeric properties in order to perform the evaluation. In the following example:

```
%LET X=100;
%LET Y=80;
%MACRO OUTDATA;
   %IF &X>&Y %THEN %DO;
      OUTPUT;
      RETURN;
      %END;
   %ELSE %STR(DELETE;);
   %MEND OUTDATA;
```

the condition of the %IF statement contains a logical expression, &X>&Y, which is evaluated as though the %EVAL function appeared. (The %STR function is used to enclose a string containing a semicolon. It is described below.)

The use of %EVAL resembles the use of expressions in SAS in most ways; however, four characteristics of %EVAL are different from the way SAS handles expressions.

- %EVAL performs only integer arithmetic. For example, these statements:

  ```
  %LET A=1;
  %LET B=3;
  %LET C=%EVAL(&A/&B);
  ```

  assign C a value of 0.

  If you attempt to use %EVAL to perform a calculation on noninteger values, the macro processor issues an error message. Thus, although the values of PI and R in this example are acceptable to the macro language:

```
%LET PI=3.14;
%LET R=5;
```

you cannot use the statement

```
%LET C=%EVAL(2*&PI*&R);
```

to calculate a value for C.

- The %EVAL function does not allow the use of SAS operators for concatenation (| |), minimum (><), or maximum (<>).
- The mathematical symbol for "greater than or equal" must be >=, since the symbols '>=' and '>=' are not equivalent in the macro language. The same restrictions apply to the mathematical symbol for "less than or equal," which must appear as '<='.
- Expressions containing two sets of operators, such as 10<&X<20, are not allowed.

A list of the operators available to the %EVAL function, their symbols, and their mnemonic forms follows. The order in which operations are performed in the macro language is the same as in SAS. As in SAS, operations within parentheses are performed first.

Table 19.2 Macro Language Operators

OPERATOR	MNEMONIC	NAME	PRECEDENCE
**		exponentiation	1
¬	NOT	logical not	2
*		multiplication	3
/		division	3
+		addition	4
−		subtraction	4
<	LT	less than	5
<=	LE	less than or equal	5
=	EQ	equal	5
¬=	NE	not equal	5
>	GT	greater than	5
>=	GE	greater than or equal	5
&	AND	logical and	6
|	OR	logical or	7

Using %EVAL in counting applications If the value of a condition is numeric, you can increment the value of the condition with a %LET statement that uses the %EVAL function, as in this example:

```
%MACRO ITER;
   %LET A=1;
   %DO %WHILE(&A<=10);
      %PUT ITERATION &A;
      %LET A=%EVAL(&A+1);
      %END;
   %MEND ITER;
```

The call %ITER writes the text shown in **Output 19.23** on the SAS log.

Output 19.23 %EVAL Function

```
11          %ITER
ITERATION 1
ITERATION 2
ITERATION 3
ITERATION 4
ITERATION 5
ITERATION 6
ITERATION 7
ITERATION 8
ITERATION 9
ITERATION 10
```

You can increment a value in constant text by decimal fractions. Using concatenation:

```
%MACRO LINES;
   %DO X=1 %TO 2;
       %DO Y=0 %TO 8 %BY 2;
          %LET X2=&X..&Y;
          &X2
          %IF &X=2 %THEN %GO TO LABEL1;
          %END;
    %LABEL1: %END;
    %MEND;
```

Macro LINES creates X2 by concatenating the value of X, the value of Y, and a decimal point so that X2 can take on values such as 1.0, 1.2, and 1.4. (See **Resolving Macro Variable References** for information on concatenating macro variable references, including the use of two decimal points in the value of X2.) Notice that allowing the Y-loop to execute five times when X is 2 produces values from 2.0 to 2.8. The %IF statement in the Y loop is executed the first time that X=2 (when Y=0); thus the last value that X2 receives is 2.0. This example uses macro LINES in a PROC PLOT statement:

```
OPTIONS NOMPRINT SYMBOLGEN;
PROC PLOT;
   PLOT Y*X/VREF=%LINES
      HREF=%LINES;
RUN;
```

SAS sees the statements in **Output 19.24**.

Output 19.24 Creating a Decimal Value in Constant Text

```
10          OPTIONS NOMPRINT SYMBOLGEN;
11          PROC PLOT;
14         +1.0                                                    3-&SYMBOL
18         +                                                1.2    3-&SYMBOL
22         +                                                1.4    3-&SYMBOL
26         +                                                1.6    3-&SYMBOL
30         +                                                1.8    3-&SYMBOL
34         +                                                2.0    3-&SYMBOL
12             PLOT Y*X / VREF=%LINES
39         +1.0                                                    3-&SYMBOL
43         +                                                1.2    3-&SYMBOL
47         +                                                1.4    3-&SYMBOL
51         +                                                1.6    3-&SYMBOL
55         +                                                1.8    3-&SYMBOL
59         +                                                2.0    3-&SYMBOL
37                HREF=%LINES;
62          RUN;
```

Using %EVAL in comparisons Although %EVAL does not perform mathematical operations on noninteger values, you can use noninteger values in comparisons. For example,

```
%MACRO TEST(BIG,LITTLE);
   %IF &BIG>&LITTLE %THEN %PUT &BIG IS BIGGER;
   %ELSE %IF &BIG=&LITTLE %THEN %PUT &BIG=&LITTLE;
   %ELSE %PUT &LITTLE IS BIGGER;
   %MEND TEST;
```

The calls in **Output 19.25** produce the results shown.

Output 19.25 Implied %EVAL Function in Comparisons

```
8 IS BIGGER
8.00 IS BIGGER
9
10          %TEST(8,5.62)
14          %TEST(8,8.00)
```

The macro processor compares character strings character by character beginning at the left until it reaches the end of the strings or until it reaches a pair of characters that do not match. If one string contains fewer characters than the other, the shorter string is padded with trailing blanks to make the lengths equal. If a character string is compared with an integer, the integer remains a character string to make the comparison, instead of being assigned numeric properties as it is for the evaluation of mathematical expressions.

Thus, when you call macro TEST with the values 8 and 5.62, the condition &BIG>&LITTLE is true because the comparison is made between the characters 8 and 5. When you use the values 8 and 8.00, the first character of each string matches, and the comparison is made between the padded blank after the 8 and the period in 8.00. Since a period is larger than a blank, 8.00 is reported as larger than 8 with trailing blanks. (See the SORT procedure description for the comparison sequence used.)

%INDEX: finds the first occurrence of a character string

%INDEX(*argument1*,*argument2*)

Argument1 and *argument2* can be character strings, macro variable references, expressions, or macro calls. The %INDEX function searches *argument1* for the first occurrence of the string identified in *argument2*. If the string is not found, the function returns a 0. The %INDEX function may be used inside and outside a macro definition. This example uses %INDEX to calculate the value of B since the character V appears in the value of A at position 3:

```
%LET A=A VERY LONG VALUE;
%LET B=%INDEX(&A,V);
%PUT V APPEARS AT POSITION &B;
```

The text produced by the %PUT statement is shown in **Output 19.26**.

Output 19.26 %INDEX Function

```
V APPEARS AT POSITION 3
```

This example shows %INDEX as part of an expression within a macro:

```
%LET NAME=SAVE.DATA;
%MACRO CHECK;
   %IF %INDEX(&NAME,SAVE)¬=1 %THEN %LET NAME=SAVE.DATA;
   SET &NAME;
   %MEND CHECK;
```

Macro CHECK is used to see if the character string SAVE is the first part of the value of macro variable NAME. If not, the %LET statement sets the value of name to SAVE.DATA; then the SET statement is issued.

%LENGTH: finds the length of an argument

%LENGTH(*argument*)

Argument can be a macro variable value, another macro function, or a macro call. The %LENGTH function returns the length of *argument*; if *argument* has a null value, %LENGTH returns a value of 0. If you use a macro call as the argument of %LENGTH, %LENGTH returns the length of the result of the macro, not the length of the macro definition. The %LENGTH function works both inside and outside macros. The following example illustrates the use of %LENGTH:

```
%MACRO LONG(C);
   DATA NEW;
      INFILE IN;
      INPUT X Y Z;
   %IF %LENGTH(&C)<=200 %THEN %DO;
      CORRECT=SYMGET('C');
      LABEL CORRECT='FROM SYMGET(''C'')';
      %END;
   %ELSE %DO;
      CORRECT=' ';
      LABEL CORRECT='TRUNCATION PROBLEM';
      %END;
```

```
    RUN;
    %MEND LONG;

%LET LIST=...a long value...;

%LONG(&LIST)
```

In this example, you assign the value of macro variable C to the DATA step variable CORRECT if that value can be supplied without truncation (see a description of the DATA step function SYMGET, below). If the value of C is too long to be passed without truncation (more than 200 bytes), you assign a missing value to CORRECT.

NRBQUOTE: removes the meaning from unanticipated special characters including & and % during macro execution

%NRBQUOTE(*argument*)

The %NRBQUOTE (**N**o **R**escan **B**lind **Quote**) function is identical to the %BQUOTE function except that it also removes the significance of the ampersand (&) and percent (%) signs, including the percent sign attached to macro functions.

%NRQUOTE: removes meaning from % and & during macro execution

%NRQUOTE(*argument*)

Argument can be any character string, including macro variable references and macro calls.

The %NRQUOTE function quotes resolved macro variable references and macro calls during macro execution (as opposed to the %NRSTR function, which takes effect while the macro is being compiled, or read, by the macro processor).

In the following interactive example, the %NRQUOTE function in macro SHOWNRQ prevents the macro processor from reading the percent sign combined with the letters abc as the macro call %abc; the %QUOTE function in macro SHOWQ allows the macro processor to recognize the percent sign combined with the letters abc as the macro call %abc. See **Output 19.27**.

Output 19.27 %NRQUOTE Function

```
  1? %macro abc;
  2?    %put From macro abc;
  3?    %mend;
  4? %macro shownrq;
  5?    %put Enter a value.;
  6?    %input;
  7?    %let p1=%nrquote(&sysbuffr);
  8?    %put &p1.abc;
  9?    %mend;
 10? %macro showq;
 11?    %put Enter a value.;
 12?    %input;
 13?    %let p2=%quote(&sysbuffr);
 14?    %put &p2.abc;
 15?    %mend;
 16? %shownrq
Enter a value.
%%
%abc
 17? %showq
Enter a value.
%%
From macro abc
 18?
```

See **Quoting in the Macro Facility** for information on the use of two percent signs, and see **Resolving Macro Variable References** for information on concatenating macro variable references and character strings.

%NRSTR: removes meaning from special characters including % and & at macro compilation

%NRSTR(*argument*)

Argument can be any character string.

The %NRSTR (**N**o **R**escan **STR**ing) function is identical to the %STR function except that it also removes the significance of the ampersand (&) and percent sign (%), including the percent sign attached to macro functions. Thus, macro calls in the argument of the %NRSTR function are not executed, macro variable references are not resolved, and macro function arguments are not evaluated.

For example, suppose you have a SAS program containing two macros named DAILY and WEEKLY. The statement

```
%PUT ENTER %NRSTR(%DAILY) OR %NRSTR(%WEEKLY).;
```

results in the message

```
ENTER %DAILY OR %WEEKLY.
```

If you do not enclose the macro calls in the %NRSTR function, the result of the %PUT statement is

```
ENTER text produced by macro DAILY OR text produced by macro WEEKLY.
```

You can enclose the entire text of the %PUT statement in the %NRSTR function for clarity; for example,

```
%PUT %NRSTR(AN EXAMPLE USING %EVAL IS %LET X=%EVAL(8*6););
```

produces the message

```
AN EXAMPLE USING %EVAL IS %LET X=%EVAL(8*6);
```

%QUOTE: removes meaning from a string (except % and &) during macro execution

%QUOTE(*argument*)

Argument can be any character string, including macro variable references and macro calls.

The %QUOTE function removes the meaning of special characters in the value of a macro variable after the macro variable reference has been resolved or in the result of a macro after the macro has been called. Its effect occurs while the macro is being executed (as opposed to the %STR function, which takes effect while the macro is being compiled, or read, by the macro processor). The %QUOTE function does not remove the meaning from the percent sign or the ampersand; to do that, use the %NRQUOTE function.

For example, suppose you create the following macro to construct a KEEP statement:

```
%MACRO KEEPIT(KLIST);
   %IF &KLIST¬= %THEN
      KEEP &KLIST %STR(;);
%MEND KEEPIT;
```

Two possible calls are

```
%KEEPIT(HEIGHT WEIGHT)
%KEEPIT(YEAR1-YEAR10)
```

In the first macro call, the %IF condition is true and the %THEN statement produces a KEEP statement:

```
KEEP HEIGHT WEIGHT;
```

However, in the second macro call, the macro processor attempts to subtract YEAR10 from YEAR1 and produces an error message, since those character strings are not valid numbers. To avoid this problem, enclose the text to the left of the equal sign in the %QUOTE function as follows:

```
%MACRO KEEPIT(KLIST);
   %IF %QUOTE(&KLIST)¬= %THEN
     KEEP &KLIST %STR(;);
%MEND KEEPIT;
```

The %QUOTE function removes the special meaning from characters in the value of KLIST after the value has been substituted into the macro. Thus, the hyphen in the string YEAR1-YEAR10 is not considered to be a subtraction operator. The %IF condition is evaluated as a comparison between the value of KLIST and a null string.

An alternative to using the %QUOTE function in this case is to use the %STR function in the macro call, as in

```
%KEEPIT(%STR(YEAR1-YEAR10))
```

However, the %QUOTE function may be more convenient, since that can be specified at the creation of the macro; the %STR function must be given in the macro call each time it is needed.

If the argument of the %QUOTE function contains an unmatched parenthesis, single quote, or double quote, put a percent sign in front of the symbol. If a percent sign in text occurs next to a parenthesis, single quote, or double quote, write the percent sign as two percent signs (%%). See **Special Characters in Macro Language Quoting Functions** later in this chapter for more information.

In general, if the argument to be enclosed in a quoting function contains macro variable references or macro calls, use the %NRSTR or %NRQUOTE function; if the argument does not contain either of these items, use the %STR or %QUOTE function.

%SCAN: scans for "words"

%SCAN(*argument,n*[,*delimiters*])

Argument, *n*, and *delimiters* can be character strings, macro variable references, or macro calls. In addition, *n* can be an expression; an implied %EVAL gives *n* numeric properties. %SCAN returns the *n*th "word" of *argument*, where "words" are strings of characters separated by one or more delimiters. If you do not specify delimiters, %SCAN uses the same list of delimiters as the DATA step SCAN function. If the number of words in *argument* is less than *n*, the result of the function is a null string. %SCAN works both inside and outside macros. For example,

```
%LET A=A//-VALUE.WITH$DELIMITERS;
%LET B=%SCAN(&A,3);
```

assigns B a value of WITH, the third "word" in the value of symbolic variable A. However, the statements

```
%LET C=EFGHAIJK/LMNOAPQ.RSTAUVW XYZ;
%LET D=%SCAN(&C,3,A);
```

assign D a value of PQ.RST, the third "word" in C (delimited by the letter A).

%STR: removes meaning from special characters (except % and &) at macro compilation

%STR(*argument*)

Argument can be any character string.

Just as you use quotes in SAS processing to surround a character string that contains special characters, the %STR function is used in the macro language to "surround" a character string containing characters special to the macro language.

For example, you may want to use a %LET statement to create a macro variable whose value represents a complete SAS statement or several SAS statements. Since the %LET statement itself ends in a semicolon the first semicolon in the value must not signal the end of the %LET statement. A semicolon enclosed by the %STR function is treated by the macro processor as part of the character string. For example,

```
%LET PRINT1=PROC PRINT %STR(;);
```

The macro processor treats the semicolon enclosed in the %STR function like the other characters in the value of PRINT1; the semicolon not enclosed in %STR ends the %LET statement. Note that the entire value of PRINT1 is a character string to the macro language and can be enclosed in the %STR function for clarity:

```
%LET PRINT1=PROC %STR(PRINT;);

%LET PRINT1=%STR(PROC PRINT;);
```

You cannot use %STR to avoid invoking the meaning of the ampersand (&), percent sign (%), or macro functions. For example,

```
%LET A=THIS VALUE;
%LET B=%STR(&A);
```

assigns B a value of THIS VALUE just as if you had specified

```
%LET A=THIS VALUE;
%LET B=&A;
```

To remove the meaning of the ampersand and percent sign (including the percent sign attached to macro functions), use the %NRSTR function discussed earlier in this section.

You can use %STR to prevent the macro processor from evaluating a macro expression in a %IF, %DO %WHILE, or %DO %UNTIL statement. For example:

```
%MACRO SHOWXY;
   %LET X=1;
   %LET Y=2;
   %IF &X+&Y=&Y+&X %THEN %DO;
      %*THIS EXPRESSION IS TRUE;
      %*TREATED AS IF %EVAL FUNCTION WERE SPECIFIED;
      %END;
   %IF %STR(&X+&Y)=%STR(&Y+&X) %THEN %DO;
      %*THIS EXPRESSION IS NOT TRUE UNLESS X AND Y HAVE
         IDENTICAL VALUES;
      %*%STR PREVENTS NUMERIC EVALUATION;
      %END;
   %MEND SHOWXY;
```

Since the expression in the first %IF statement is evaluated using the rules of the %EVAL function, the expression is true for any numbers. The expression 3=3 is evaluated. In this case the expression in the second %IF statement is true only if the values of X and Y contain the same characters in the same order, since the use of %STR on each side of the expression causes the expression to read "the character string on the left equals the character string on the right." In this case, the expression 1+2=2+1 is evaluated.

If an expression contains a character string with a word that is also an arithmetic or logical operator, you should enclose the character string in the %STR function so that the expression is not evaluated as if a %EVAL function were present.

If the argument of the %STR function contains an unmatched parenthesis, single quote, or double quote, put a percent sign in front of the symbol. If a percent sign in text occurs next to a parenthesis, single quote, or double quote, write the percent sign as two percent signs (%%). See **Special Characters in Macro Language Quoting Functions** later in this chapter for more information.

%SUBSTR: substrings a character string

%SUBSTR(*argument,position*[,*length*])

Argument, *position*, and *length* can be character strings, macro variable references, expressions, macro functions, or macro calls. %SUBSTR produces a substring of *argument* beginning at *position* for a length of *length* characters. If you do not supply a value for *length*, %SUBSTR produces a string containing the characters from *position* to the end of *argument*. %SUBSTR works both inside and outside macros. For example:

```
%LET FIRST=THIS VALUE;
%LET SECOND=%SUBSTR(&FIRST,6,5);
```

SECOND has a value of VALUE, the result of selecting a character string from the value of FIRST, beginning at position 6 for a length of 5.

```
%LET THIRD=%SUBSTR(&FIRST,%INDEX(&FIRST,H),2);
```

THIRD has a value of HI, the result of selecting a character string from the value of FIRST, beginning at the first occurrence of H and having a length of 2.

```
%LET FOURTH=%SUBSTR(&FIRST,%INDEX(&FIRST,H));
```

FOURTH has a value of HIS VALUE, the result of selecting a character string from the value of FIRST beginning at the first occurrence of H and continuing to the end of the value of FIRST.

%UNQUOTE: undoes quoting

%UNQUOTE(*argument*)

Argument may be any character string, possibly containing macro calls or macro variable references.

The %UNQUOTE function undoes the effect of %STR, %NRSTR, %BQUOTE, %NRBQUOTE, %QUOTE, and %NRQUOTE. That is, it restores the special meaning that was removed from characters by one of these functions. It takes effect during macro execution. For example, in this system of macros:

```
%MACRO SECURE;
   %PUT THE NIGHT SECURITY CODE IS 84629.;
%MEND SECURE;
```

```
%MACRO ACCESS(PW) / STMT;
   %LET INFO =%NRSTR(RESTRICTED INFORMATION. %SECURE);
   %IF %UPCASE(&PW)=MYPASSWORD %THEN %PUT %UNQUOTE(&INFO);
   %ELSE %PUT &INFO;
%MEND ACCESS;
```

the two calls

```
ACCESS;
ACCESS MYPASSWORD;
```

produce different results, as shown in **Output 19.28**.

Output 19.28 %UNQUOTE Function

```
66          ACCESS;
RESTRICTED INFORMATION. %SECURE
71          ACCESS MYPASSWORD;
THE NIGHT SECURITY CODE IS 84629.
RESTRICTED INFORMATION.
```

In the first call, the value of INFO includes the characters %SECURE, not the result of macro SECURE. Since the second call contains the correct value for PW, the %PUT statement writes the unquoted value of INFO, or the result of calling macro SECURE. (The %UPCASE function causes the value of PW to be translated to uppercase before being compared to the uppercase string MYPASSWORD. See the %UPCASE function for more information.)

%UPCASE: translates lower case characters to upper case

%UPCASE(*argument*)

Argument can be any character string, including macro variable references and macro calls.

The %UPCASE function translates lowercase characters in *argument* into uppercase. The function is useful in comparisons because the macro facility does not automatically translate lowercase characters to uppercase before performing a comparison. For example, using the %UPCASE function in this macro:

```
%MACRO SHORT;
   %PUT ENTER A THREE-CHARACTER MONTH NAME;
   %INPUT MONTH;
   %IF %UPCASE(&MONTH)=DEC %THEN %STR(PROC PRINT DATA=ENDYR;);
   %ELSE %STR(PROC CONTENTS;);
   %MEND SHORT;
```

causes all three responses:

```
DEC
Dec
dec
```

to satisfy the condition in the %IF statement.

DATA Step Functions Used with the Macro Language

Since the functions described here belong to the DATA step, the DATA step rules for quoting the argument of a function apply. That is, an argument enclosed in

quotes is a character string; an argument not enclosed in quotes is the name of a SAS variable in the same DATA step.

SYMGET: returns the value of a macro variable

SYMGET(*argument*)

Argument can be the name of a DATA step character variable or a character string representing the name of a macro variable.

You can use the SYMGET function in a DATA step to create a SAS variable whose values are macro variable values. The value returned by SYMGET is the value of a macro variable identified by *argument*. If *argument* is a DATA step variable name, the value of the DATA step variable must be the name of a macro variable. For example:

```
%LET A1=BEGIN;
%LET A2=END;
%LET A3=CONTINUE;
DATA ONE;
   INPUT CODE $;
   KEY=SYMGET(CODE);
   CARDS;
A1
A1
A2
A3
A1
A2
;
PROC PRINT;
```

In DATA step ONE, SAS first reads a character variable CODE with values of A1, A2, or A3. (Each value of CODE must conform to the rules for a SAS name and that each is a macro variable name.) Each time the assignment statement is executed, the SYMGET function selects the macro variable whose name is the current value of CODE and assigns the value of that macro variable as the value of KEY for the current observation. The result of the PRINT procedure is shown in **Output 19.29**.

Output 19.29 SYMGET Function

SAS 1

OBS	CODE	KEY
1	A1	BEGIN
2	A1	BEGIN
3	A2	END
4	A3	CONTINUE
5	A1	BEGIN
6	A2	END

If the value of CODE for an observation does not conform to the rules for a SAS name, or if SAS cannot find a macro variable of that name, SAS prints a warning message and sets the resulting value to missing.

In this example *argument* is a character string representing the name of a macro variable:

```
%LET G=GOOD;
DATA TEST2;
   SET TEST1;
   X=SYMGET('G');
RUN;
```

In each observation of data set TEST2, X has the value GOOD.

The SYMGET function produces a character value, and a variable you create in an assignment statement using SYMGET is a character variable (unless you have previously established the variable as numeric, for example with a LENGTH statement). If the value of the macro variable is more than 200 characters (the maximum length of a DATA step character variable), SAS truncates the value of the macro variable on the right. If the macro variable value represents a number, as in this example:

```
%LET START=3;
```

you can use SYMGET('START') in calculations, since SAS automatically converts a character value used in a calculation to a numeric value. The following example illustrates several uses of SYMGET:

```
%LET START=3;
DATA NEW;
   SET OLD;
   A=SYMGET('START');
   LENGTH B 8;
   B=SYMGET('START');
   C=SYMGET('START')/4;
   D=B/4;
RUN;
```

A is a character variable with a value of '3' and a length of 200 (the 3 is padded with trailing blanks to make 200 characters). B is a numeric variable with a length of 8, since B is defined by the LENGTH statement as numeric. SAS automatically converts the character value of SYMGET('START') to a numeric value when it assigns the value to B.

To create C, the SYMGET function is used in a calculation, and for D the same calculation uses a variable created from SYMGET('START') instead of the function itself.

SYMPUT: assigns a value from a DATA step to a macro variable

CALL SYMPUT(*argument1*,*argument2*);

You can use the SYMPUT function to create one or more macro variables whose value is the value of a DATA step variable. Thus, you can use the SYMPUT function to pass values from one DATA step to another DATA or PROC step in a SAS program.

In the SYMPUT function, *argument 1* identifies the name of the macro variable. *Argument 2* contains the value to be assigned to the macro variable. You must use the DATA step statement CALL with SYMPUT. (See "Statements Used in the DATA Step" for a discussion of the CALL statement.)

The form of *argument1* can be

- the name of a macro variable. Enclose the name in quotes. For example,

  ```
  CALL SYMPUT('NEW',argument2);
  ```

  The result is a macro variable named NEW with a value of *argument2*. If you have not previously created NEW, the SYMPUT function creates it in

the current referencing environment. If a variable named NEW already exists in the current referencing environment, the SYMPUT function assigns it the value of *argument2*. In either case the SYMPUT function affects one macro variable.

- the name of a DATA step character variable whose value in the current observation is the name of the macro variable to be used. Do not enclose the name of the DATA step variable in quotes. For example:

```
DATA A;
   INPUT PERSON $;
   CALL SYMPUT(PERSON,argument2);
   CARDS;
ANN
TOM
BILL
;
```

The SYMPUT function creates three global macro variables named ANN, TOM, and BILL. Each of them has the value of *argument2* for that observation. If any of these variables existed previously, they receive new values through the SYMPUT function.

The form of *argument2* can be

- a character string to be assigned as the value of all macro variables created by the SYMPUT function. Enclose the character string in quotes. For example, the statements

```
CALL SYMPUT(argument1,'TEST VALUE');
```

assign the character string TEST VALUE to all macro variables created by *argument1*.

- the name of a DATA step variable whose value in the current observation is assigned to the macro variable. The variable can be numeric or character; do not enclose the name of the DATA step variable in quotes. For example:

```
DATA B;
   INPUT CITY $;
   CALL SYMPUT(argument1,CITY);
   CARDS;
RALEIGH
CARY
APEX
;
```

Each time the CALL SYMPUT statement is executed, the value of CITY for the current observation is assigned to the macro variable named in *argument1*. When the DATA step has more than one observation, this form is most useful when *argument1* is also the name of a SAS variable, since a unique macro variable name and value are created from each observation. If *argument1* is a character string and *argument2* is the name of a DATA step variable, multiple observations simply change the value of the macro variable; only the value from the last observation is preserved after the end of the DATA step.

- the name of a DATA step function. The value returned by the function in the current observation is assigned as the value of the macro variable in *argument1*. Do not enclose the DATA step function in quotes. For example:

```
DATA C;
   INPUT X MMDDYY.;
   CALL SYMPUT(argument1,PUT(X,WORDDATE.));
   CARDS;
070485
;
```

The macro variable identified in *argument1* receives the value JULY 4, 1985.

If *argument2* is a number, the SYMPUT function "writes" it using the BEST12. format. The value of the macro variable is a twelve-byte character string with the value right-aligned within it. For example,

```
DATA SHOW1;
   X=1;
   CALL SYMPUT('TESTVAR',X);
RUN;
%PUT ***&TESTVAR***;
```

produces **Output 19.30**.

Output 19.30 Right-aligned Macro Variable Value Produced by SYMPUT Function

```
93            DATA SHOW1;
94               X=1;
95               CALL SYMPUT('TESTVAR',X);
96            RUN;

NOTE: NUMERIC VALUES HAVE BEEN CONVERTED TO CHARACTER
      VALUES AT THE PLACES GIVEN BY: (LINE):(COLUMN).

      95:29

NOTE: DATA SET WORK.SHOW1 HAS 1 OBSERVATIONS AND 1 VARIABLES. 1588 OBS/TRK.
97            %PUT ***&TESTVAR
98         +             1                                                  1-&SYMBOL
97                             ***;
***           1***
```

If you use the value of TESTVAR in a SAS operation that does not trim leading blanks, the right alignment can produce unexpected results. For example, the statements

```
DATA SHOW2;
   LENGTH RESULT $10;
   RESULT='ABC'||&TESTVAR;
PROC PRINT;
TITLE "STUDY OF &TESTVAR";
RUN;
```

produce **Output 19.31**.

Output 19.31 Using Right-aligned Value Produced by SYMPUT

```
STUDY OF          1                                                    2

  OBS    RESULT

    1      ABC
```

Since the length of variable RESULT is 10, RESULT is composed of the three characters ABC and the first seven characters of &TESTVAR—all blanks. The TESTVAR is truncated before the 1. In addition, the character 1 appears in the title preceded by leading blanks. To eliminate the leading blanks, use the PUT function to assign a format or the LEFT function to left-align the result of SYMPUT. The following example eliminates the problems in the previous example:

```
DATA SHOW3;
   X=1;
   CALL SYMPUT('TESTVAR',LEFT(X));
DATA SHOW4;
   LENGTH RESULT $10;
   RESULT='ABC'||"&TESTVAR";
PROC PRINT;
   TITLE "STUDY OF &TESTVAR";
RUN;
```

The result of the PROC PRINT step is shown in **Output 19.32.**

Output 19.32 Left-aligning Value Produced by SYMPUT

```
                STUDY OF 1                                             3

  OBS    RESULT

    1      ABC1
```

In this example, the value identified in *argument2* is assigned to the the macro variable named *argument1*. The statements

```
DATA NEW;
   INPUT ID $ X1-X10;
   N+1;
   CALL SYMPUT('NOBS', N);
   CARDS;
data lines
;
PROC PRINT;
   TITLE "LISTING OF ALL &NOBS OBSERVATIONS";
```

create macro variable NOBS, whose value at the end of the DATA step is the number of observations in the data set. The macro variable NOBS can then be referenced in the TITLE statement.

The macro below can be used to check for input data errors:

```
%MACRO CHECK(VARLIST);
   %LET ERRCK=NO;
   DATA NEW ERRDATA;
      INFILE IN;
      INPUT &VARLIST;
      IF _ERROR_=1 THEN DO;
         CALL SYMPUT('ERRCK','YES');
         OUTPUT ERRDATA;
         END;
      OUTPUT NEW;
      RUN;
   %IF &ERRCK=YES %THEN %DO;
      PROC PRINT DATA=ERRDATA;
         VAR &VARLIST;
      %END;
   %MEND;
```

If SAS finds an error in your data and sets the automatic variable _ERROR_ to 1, the value of macro variable ERRCK is given the value YES. The observation in error is written to SAS data set ERRDATA. When the DATA step is complete, if an error was found and ERRCK's value was YES, SAS prints the data set ERRDATA.

Automatic Macro Variables

The macro language includes some automatic macro variables which can be referenced anywhere in your SAS job by preceding their name with an ampersand (&). All variables begin with SYS and are global. You use them exactly as you do other macro variables that you create. Caution: avoid naming your own macro variable names beginning with the letters SYS in order to avoid conflicts with current or future SAS-defined macro variables.

SYSBUFFR receives excess text entered in response to a %INPUT statement. For example, if the %INPUT statement has the following form:

```
%INPUT NAME CITY;
```

and you enter

```
SMITH CARY NC
```

the three characters *' NC'* become the value of SYSBUFFR because the characters ' NC' do not correspond to any of the macro variables in the %INPUT statement (see **Macro Program Statements** for more information). Until the first %INPUT statement is executed, SYSBUFFR has a null value. SYSBUFFR receives a new value each time a %INPUT statement is executed: either the excess characters entered in response to the %INPUT statement or a null value if no excess characters are present. If the %INPUT statement contains no macro variable names, all characters entered are assigned to SYSBUFFR.

SYSDATE gives the date (in DATE6. or DATE7. format) on which the SAS job started execution, for example, 2JUN85.

SYSDAY	gives the day of the week the SAS job started execution. SYSDAY appears as a word, for example, MONDAY.
SYSDEVIC	gives the name of the current graphics device, the value of the SAS system option DEVICE=.
SYSDSN	gives the name of the last SAS data set created.
SYSENV	returns FORE (as in foreground) if you are working in display manager or interactively and BACK (as in background) if you are not.
SYSINDEX	gives the number of macros that have started execution so far in the current SAS job.
SYSPARM	returns the same string as that returned by the DATA step function SYSPARM().
SYSRC	gives the last return code set by the macro language. SYSRC is determined by the macros and the macro language service routines.
SYSSCP	returns the abbreviation for the operating system you are using. Values of SYSSCP can be CMS, DOS, OS.
SYSTIME	gives the time the SAS job started execution, as in 15:25.
SYSVER	gives the version of the SAS System you are using, as in 5.01.

SPECIAL TOPICS

Resolving Macro Variable References

Replacing a macro variable reference with the value of the macro variable is known as *resolving a macro variable reference* or *symbolic substitution.*

Simple macro variable references In this example:

```
%LET E=ENTER;
%PUT &E THE VARIABLE NAMES;
```

resolving &E and executing the %PUT statement yields:

```
ENTER THE VARIABLE NAMES
```

The macro processor resolves a macro reference by

- reading the ampersand and the "word" following it for 8 characters (the maximum length of a SAS name) or until a character that cannot be part of a SAS name is encountered. In the example above, the space character, which cannot be part of a SAS name, ends the reference &E.
- searching its macro variable storage areas (called *symbol tables*) for the variable. If it finds the variable, it replaces the reference with the value (including a null value, or 0 characters, if necessary); if it does not find the variable, it leaves the reference unresolved.

The most common causes of an unresolvable reference are referring to the variable before you define it and misspelling the variable name. If the macro variable reference passes unresolved into a SAS statement, the reference usually causes a SAS syntax error (because few SAS statements accept the ampersand). How-

ever, at times you may want to leave a macro variable reference temporarily unresolved for further use within the macro facility. In addition, macro variable references within single quotes in SAS statements are not resolved; macro variable references within double quotes are resolved if the DQUOTE option is in effect. (See **SAS Quoting and the Macro Facility** for more information.) Macro variable references in SAS comment statements are not resolved.

Indirect macro variable references You can use multiple ampersands to refer to the value of a macro variable indirectly. For example, when the macro processor encounters the indirect reference

```
&&&E
```

as in

```
%LET E=ENTER;
%LET ENTER=ENTER ONE OF;
%PUT &&&E THE VARIABLE NAMES;
```

it examines the first ampersand and the "word" following it (in this case, another ampersand, since a special character can constitute a "word," or token). The macro processor always resolves two ampersands into a single ampersand. The macro processor then examines the next (third) ampersand and the following word and replaces &E with ENTER. The reference at this stage is &ENTER. The macro processor does an additional scan of all references that were changed in the previous scan and resolves &ENTER to ENTER ONE OF. Executing the %PUT statement then produces

```
ENTER ONE OF THE VARIABLE NAMES
```

You can use any odd number of ampersands to create an indirect macro variable reference; the macro resolves them by pairing ampersands beginning at the left and making repeated scans, as in this example. Indirect macro variable references can be useful in concatenations (below).

Concatenating macro variable references You can concatenate macro variable references to other macro variable references and to character strings. For example, both of these %PUT statements:

```
%LET X=BIRTH;
%LET Y=DAY;
%PUT HAPPY BIRTH&Y;
%PUT HAPPY &X&Y;
```

write

```
HAPPY BIRTHDAY
```

To write the text BIRTH using the reference &X and the text DAY using the string itself, you must explicitly end the macro variable reference by putting a period at the end of the macro variable name:

```
%PUT HAPPY &X.DAY;
```

The macro processor interprets the period as a delimiter ending the macro variable reference and does not print it. The result is again

```
HAPPY BIRTHDAY
```

(Omitting the period causes the macro processor to search for the macro variable XDAY because XDAY is a valid SAS name.)

To produce a period in text following a macro variable reference, code two periods. For example, the statement

```
%PUT HAPPY &X&Y..;
```

yields

```
HAPPY BIRTHDAY.
```

You can combine indirect macro variable references, concatenation, and delayed resolution of macro variable references to produce complete text strings. For example, these statements:

```
%LET X=BIRTH;
%LET Y=DAY;
%LET BIRTHDAY=ANNIVERSARY;
%PUT HAPPY &&&X&Y...;
```

write

```
HAPPY ANNIVERSARY.
```

In the resolution, the macro processor resolves the first two ampersands, the reference &X, and the reference &Y. in a single scan to produce the string

```
&BIRTHDAY..
```

The macro processor then resumes scanning the string, beginning with a rescan of all text that was altered in the preceding scan. Since the string now contains the reference &BIRTHDAY., the text produced is

```
HAPPY ANNIVERSARY.
```

Use the SYMBOLGEN option to see the intermediate strings.

See **Examples** at the end of this chapter for more examples of resolving macro variable references.

Referencing Environments

SAS jobs that contain macro processor activity have *referencing environments*. A referencing environment is the area in which a macro variable or a group of macro variables is available for use. Referencing environments are important because to transfer information from one macro variable to another, from a macro variable to constant text, or from a macro variable to DATA step execution or vice versa, the macro variables must be available in the same referencing environment.

The largest possible referencing environment is the global environment, that is, the entire SAS job. The global environment contains SAS statements outside of any macro, macro variables created outside of any macro, and macro variables created in %GLOBAL statements. The global environment and the macro variables in it exist for the duration of the SAS job or session. A macro that is defined in the global environment and that does not create any macro variables except in %GLOBAL statements also executes in the global environment.

A macro which creates a macro variable other than in a %GLOBAL statement creates a local referencing environment, and any macro nested within that macro which creates a macro variable (except in a %GLOBAL statement) creates another level of local referencing environment. Macro variables created within macros include macro parameters, macro variables created as index variables in %DO statements, and macro variables created in %INPUT, %LET, and %LOCAL statements. A %GOTO statement that creates a label containing a macro variable reference or a macro call also creates a local referencing environment.

A macro variable is available in the environment in which it is defined and in all environments nested within that environment ("farther in"). You can block the availability of a macro variable in an environment by creating a local macro vari-

able of the same name at that level. The new variable blocks the availability of the original variable in that environment and is available in the environment in which you created it and in all environments more deeply nested.

DATA and PROC steps can execute in any referencing environment. The environment in which a particular DATA or PROC step executes is the one in which SAS determines the step is complete. That is, a macro which has created a local environment and contains a complete SAS step causes SAS to execute that step within its local environment. You can change the environment in which a step executes by controlling the end of the step (for example, by placing a RUN statement inside or outside of a macro). The main point for which you need to consider the environment of a DATA step is in using the SYMPUT and SYMGET functions, since the SYMPUT function creates macro variables in the current referencing environment, and a macro variable must be available to the environment of the SYMGET function in order to be retrieved.

In this example:

```
%LET A=1;
TITLE "EXAMPLE WITH A=&A";
TITLE2 "EXAMPLE WITH A GLOBAL AND A LOCAL REFERENCING ENVIRONMENT";
%MACRO OUT;
   %IF &A=1 %THEN %DO;
       %DO B=1 %TO 20 %BY 2;
          DATA DATA&B:
             INFILE IN;
             INPUT X Y Z:
             YEAR=&B;
          %END;
      %END;
   %ELSE %DO;
      %LET B=100;
      DATA DATA&B;
         SET;
      %END;
   %MEND OUT;
%OUT
```

macro variable A is defined outside any macros, that is, in the global environment. (A is referenced outside of any macro and also within macro OUT.) Macro variable B is defined in the local environment of macro OUT. A job can also contain nested referencing environments.

Quoting in the Macro Facility

Quoting text in the macro facility differs from quoting text in a SAS statement because the macro processor treats the SAS quoting characters (the single or double quote) as text. Thus, to produce the effect of "quoting" you must enclose the text to be quoted in the argument of a macro function, not in the SAS quoting characters.

Enclosing any of the quoting functions in the %UNQUOTE function reverses the effect of the quoting function.

See **Macro Functions** for descriptions and examples of individual quoting functions.

Table 19.3 summarizes the macro quoting functions.

Table 19.3 Macro Quoting Functions

Function	Removes meaning from the following characters	Takes effect during	Usually used when
%STR	all except &, %	macro compilation	argument contains constant text
%NRSTR	all	macro compilation	argument contains constant text
%QUOTE	all except &, %	macro execution	argument contains macro variables or macro calls
%NRQUOTE	all	macro execution	argument contains macro variables or macro calls
%BQUOTE	all except &, %, mnemonic operators	macro execution	argument is a macro variable value that contains unmatched special characters
%NRBQUOTE	all except mnemonic operators	macro execution	argument is a macro variable value that contains unmatched special characters

Special characters in the arguments of quoting functions Within the argument of the %STR, %NRSTR, %QUOTE, and %NRQUOTE functions:

- %' represents an unmatched single quote (')
- %" represents an unmatched double quote (")
- %(represents an unmatched left parenthesis (( )
- %) represents an unmatched right parenthesis ( ))
- %% represents a single percent sign (%) next to a parenthesis or quote.

For example, the statement

```
%PUT %STR(JILL%'S AGE);
```

writes the text

```
JILL'S AGE
```

on the SAS log.

These statements:

```
%LET FIRST=%STR(DATA _NULL_; X=3; Y=2; Z=%(X+);
%LET SECOND=%STR(Y%)/2; PUT X= Y= Z=; RUN;);
%LET THIRD=&FIRST&SECOND;
```

assign a complete DATA step as the value of THIRD. In FIRST, the % sign in front of the left parenthesis following Z= causes that parenthesis to be treated like the other characters in the string. (If you do not use a % sign in front of that parenthesis, the macro processor produces an error message for non-matching parentheses.) In SECOND, the % sign in front of the right parenthesis following Y

prevents the macro processor from reading that parenthesis as the end of the argument of %STR. The value of THIRD thus becomes:

```
DATA _NULL_; X=3; Y=2; Z=(X+Y)/2; PUT X= Y= Z=; RUN;
```

You can use the double percent sign (%%) to produce a percent sign next to a quote or parenthesis in constant text within the argument of a quoting function. For example, to write

```
RANGE 5%-10%
```

on the SAS log using a macro variable and the %STR function, code

```
%LET R=%STR(RANGE 5%-10%%);
%PUT &R;
```

In the value of R, 10%% resolves into the text 10%, and the right parenthesis closes the argument of %STR. Since the percent sign in 5% is not followed by a quote or parenthesis, you do not have to code it as a double percent sign.

To produce two percent signs in constant text, code four percent signs. For example, the statement

```
%PUT %STR(EXAMPLE OF PRODUCING TWO PERCENT SIGNS (%%%%));
```

yields

```
EXAMPLE OF PRODUCING TWO PERCENT SIGNS (%%)
```

SAS quoting and the macro facility When you use a macro variable reference or a macro call as part of SAS text that must be enclosed in quotes (such as a title or a variable label) the reference is resolved or the macro is invoked if the text is enclosed in double quotes (quotation marks); the reference is not resolved and the macro is not called if the text is enclosed in single quotes (apostrophes). Since the DQUOTE system option allows you to enclose strings in either single or double quotes, the DQUOTE option should be in effect when you use the macro facility.

For example, these statements:

```
OPTIONS DQUOTE;
FOOTNOTE1 "DATA FOR &SYSDAY USING DOUBLE QUOTES";
FOOTNOTE2 'ANALYZED ON &SYSDATE USING SINGLE QUOTES';
```

produce the following footnote lines using the automatic macro variables SYSDAY and SYSDATE:

```
FOOTNOTE1 "DATA FOR MONDAY USING DOUBLE QUOTES";
FOOTNOTE2 'ANALYZED ON &SYSDATE USING SINGLE QUOTES';
```

In addition, since SAS uses both single and double quotes in pairs, you must make sure that your macros and macro variables produce correctly paired quotes around SAS literals.

Timing of Macro Facility Actions

The SAS System divides a SAS job with no macro facility actions into two phases: *compilation* and *execution*. During compilation SAS reads SAS statements and checks them for errors. During execution SAS performs the actions indicated by the statements—reading observations, assigning values to variables, outputting observations, and so on.

Compilation and execution take place alternately: first SAS compiles a step; then it immediately executes the step. SAS recognizes the end of a step when it encounters the word DATA or PROC (that is, the beginning of a new step), the

word RUN, or a semicolon following data lines. The end of a step is called the *step boundary* and is also usually a *phase boundary*, that is, the place where SAS switches from compilation to execution and back. (In the special case in which data lines follow a CARDS or CARDS4 statement, the phase boundary occurs at the CARDS statement, where SAS stops compiling statements and begins executing the statements on the data lines; the step boundary occurs at the end of the data lines.)

In a SAS job with no macro actions, you seldom have to think about compilation versus execution or step boundaries. SAS manages them for you.[5] When a job contains macro actions, however, you must consider macro compilation and execution and how they interact with SAS compilation and execution.

The macro facility begins to compile a macro when it encounters a %MACRO statement. Macro compilation ends when the macro processor encounters a %MEND statement; however, the macro processor does not execute the macro immediately. Instead, it stores the compiled macro code in the WORK library; when you invoke the macro, the macro processor begins to execute the macro. (If you invoke the macro again, the macro processor simply executes it; the macro processor does not recompile it.)

The following example illustrates the relationship of macro processor compilation and execution and SAS compilation and execution. Match the circled numbers to the paragraphs below:

```
DATA REPORT;  ❶
   MERGE SHIPMNTS(KEEP=REPTTIME ID) PRODUCTS(KEEP=MODEL ID INVENTRY);
   BY ID;
RUN;
%MACRO TIME;  ❷
   %GLOBAL PERIOD;  ❺
   DATA _NULL_;  ❻
      SET REPORT(OBS=1);
      CALL SYMPUT('T',REPTTIME);
   RUN;
   %IF &T=MONTH %THEN %LET PERIOD=CURRENT MONTH;  ❼
   %ELSE %LET PERIOD=CURRENT YEAR;
   %MEND TIME;
*CALL MACRO TIME NOW TO CREATE MACRO VARIABLE PERIOD;  ❸
%TIME  ❹
PROC PRINT DATA=REPORT;  ❽
   TITLE "REPORT FOR &PERIOD";
RUN;
```

1. SAS scans each SAS statement for errors and compiles the statement. Since no macro actions are present, the macro processor is not triggered. SAS stops compiling statements when it encounters the RUN statement; it then executes the step.
2. After the execution of the DATA REPORT step, SAS is ready to begin compiling another step. However, the first "word" (token) encountered is a percent sign. At that point SAS stops compilation; the macro processor examines the percent sign and the word following it to see what macro activity is required. The keyword %MACRO tells the macro processor to begin compiling a macro. Thus, the macro processor compiles all material until the %MEND statement into a macro. (The SAS compiler takes no action while the macro processor is compiling a macro.)

3. After the macro processor finishes compiling the macro, SAS resumes scanning statements. The first statement SAS encounters in this example is a SAS comment statement.
4. When SAS encounters the percent sign after the comment statement, SAS stops compiling statements and the macro processor examines the percent sign and the following word to see what action to take. In this case the call %TIME indicates that the macro processor is to execute macro TIME.
5. When the macro processor begins to execute macro TIME, it first creates the global macro variable PERIOD.
6. Next the macro processor produces a piece of constant text. All constant text produced by the macro facility is immediately passed to SAS. The SAS compiler scans this text, compiles a complete DATA step, and executes it. The macro processor pauses while the SAS step is being compiled and executed. In this case, the DATA step contains a CALL SYMPUT function that produces a macro variable named T.
7. After the DATA step has executed, the macro processor executes the next macro statement, a %IF statement. The %IF statement tests the value of macro variable T (just assigned by the DATA step) and, if the value is MONTH, assigns macro variable PERIOD the value CURRENT MONTH. If T has any value other than MONTH, the %ELSE statement is executed to assign PERIOD the value CURRENT YEAR. The macro processor ceases execution.
8. SAS begins to compile statements again. When SAS encounters the ampersand, SAS stops compiling the TITLE statement; the macro processor scans the ampersand and the following word, recognizes that macro variable resolution is required, and supplies the text to replace &PERIOD. SAS resumes compiling the PROC PRINT step and, when it reaches the RUN statement, executes the step.

In summary, while macro compilation is taking place, neither SAS compilation nor SAS execution occurs. When macro execution begins, macro execution, compilation of nested macros, SAS compilation, and SAS execution can occur alternately, depending on the activity within the macro and on the SAS statements produced.

Diagnosing Errors in the Macro Facility

There are three kinds of errors in using the macro facility:

- errors in SAS statements produced by a macro
- errors in macro compilation
- errors in macro execution.

Errors in SAS statements An error in a SAS statement produced by a macro causes a SAS error message, for example:

```
ERROR 180: STATEMENT IS NOT VALID OR IS USED OUT OF PROPER ORDER.
```

An error in a SAS statement occurs when the macro processor is able to compile (read and scan for errors) the macro, but macro execution does not produce the text that you intended. To see where the error occurred, use the MPRINT option (as well as the SYMBOLGEN option if the job contains any macro variable references outside a macro). The MPRINT option displays the SAS statements as the SAS compiler sees them.

Errors in macro compilation An error in a macro that prevents it from compiling produces a four-digit error message, for example:

```
ERROR 1115: %IF STATEMENT HAS NO CONDITION.
```

The macro processor does not create a macro it cannot compile; therefore, it does not locate that macro when you invoke it. The macro call passes unresolved into the SAS program, and, if the system option MERROR is in effect, the macro processor produces the message

```
WARNING 1353: APPARENT MACRO INVOCATION NOT RESOLVED
```

To diagnose a problem in a macro that does not compile, specify

```
OPTIONS NOMPRINT MLOGIC MACROGEN SYMBOLGEN;
```

The NOMPRINT option must be in effect for the MLOGIC option to work; the MACROGEN and SYMBOLGEN options show the statements produced by a macro and macro variable substitution; and the MLOGIC option shows the places where branching occurs within a macro (such as in the condition of a %IF statement). Note: the output produced by the MLOGIC option can be large. Before using the MLOGIC option, set %DO goups to execute only a few times.

To see the text produced by the MPRINT, MLOGIC, MACROGEN, and SYMBOLGEN options, the complete SAS log must be routed to the terminal.[6]

See **Examples** for examples of the text produced by various macro printing options.

Errors in macro execution Errors in macro execution occur when the macro processor can compile the macro, but the macro does not operate as you intended. As a consequence, the macro may produce unexpected SAS statements.

The most common errors in macro execution happen when you omit an ampersand or percent sign or use one in place of the other. The problem often occurs with macro language keywords that look like SAS keywords; for example, you may accidentally code IF for %IF or %DO WHILE for %DO %WHILE. The effect of such an error depends on the context in which it occurs.

A special problem occurs when you make an error in a %MEND statement (such as typing MEND or &MEND). In this case the macro processor does not recognize the end of a macro and continues to compile all subsequent statements into the macro. If the problem occurs in a display manager or interactive session, the session seems paralyzed. To recover from this error, type %MEND statements repeatedly until you receive an error message; the error message indicates that the previous %MEND statement closed the macro and the current %MEND statement is extraneous. You can now work normally in the SAS session.

A different problem can occur in display manager or interactive sessions when you invoke a macro that contains a parameter list without supplying values for any parameters. You must put another token (such as an empty set of parentheses or a null statement) after the call. Otherwise the macro processor waits to determine whether a list of parameter values is present and does not begin to execute the macro. (Once again, the session seems paralyzed.)

Performance Considerations

The most important performance consideration to remember in using the macro processor is that you are creating SAS statements and your computer costs reflect the cost of executing those statements, exactly as though you had entered all the statements yourself. For example:

```
%MACRO CREATE;
   %DO I=1 %TO 50;
      DATA DATA&I;
```

```
   INFILE IN&I;
   INPUT X Y Z;
   RUN;
   PROC PRINT DATA=DATA&I;
   RUN;
   %END;
%MEND CREATE;
```

With macro CREATE you enter only a few statements. However the SAS statements produced create and print **fifty** data sets, which can reflect a considerable use of computer resources

To perform simple text substitution, such as placing the name of a particular month in the title of a report, put the information into the value of a macro variable instead of into a macro. If you want conditional substitution (changing the text substituted depending on various factors), if the text to be substituted contains characters that require special treatment in the argument of a quoting function, or if the text is longer than the maximum length of a macro variable value at your installation, enclose the text in a macro. In general, macro variables are faster than macros, both to create and to execute.

Macro variables are also more efficient than macros used in earlier versions of the SAS System. The simplest way to transform one of these earlier macros into a macro variable is to use a %LET statement with the name of the macro as the name of the variable and the body of the macro as the value of the variable. Be sure to insert %STR() wherever a semicolon occurs that would incorrectly end the value of the macro variable. Consider this macro from a job created under the SAS 79.6 release:

```
MACRO PR PROC PRINT; BY DEPT;%
```

If you specify

```
%LET PR=%STR(PROC PRINT; BY DEPT;);
```

then you can refer to the value of the macro variable in your job by putting an ampersand in front of the variable name. These two jobs perform the same function:

```
TITLE "EXAMPLE USING OLD MACRO";
MACRO PR PROC PRINT; BY DEPT;%
DATA NEW;
   INPUT X Y Z;
   CARDS;
data lines
;
PR

TITLE "EXAMPLE USING MACRO VARIABLES FROM MACRO LANGUAGE";
%LET PR=%STR(PROC PRINT; BY DEPT;);
DATA NEW;
   INPUT X Y Z;
   CARDS;
data lines
;
&PR
```

However, the second one is much more efficient.

If you use the system option NOMACRO at the beginning of your job, then the macro processor is unavailable for use in your job. (See "SAS System Options"

for a discussion of the NOMACRO option.) Using NOMACRO removes the availability of the macro language from SAS, including macro variables and the SYMPUT and SYMGET DATA step functions. Since macro variables, SYMPUT, and SYMGET do not invoke the entire macro processor, you may want to have the macro language available for these purposes even when you have no macros in your job.

If the IMPLMAC option is in effect and you have defined a statement-style macro, SAS checks the first word of every SAS statement to see whether it is a statement-style macro call; with name-style macros, SAS checks only for the & and % symbols. Thus, you can increase the efficiency of your SAS programs by defining statement-style macros only when you need their features and using the IMPLMAC option only if you plan to use statement-style macro invocations.

Finally, when deciding whether to use a macro in a certain situation, or whether to use a macro variable or a macro, consider the overhead cost of using macros. The cost of using macros and macro variables varies directly with the number and size of macros and macro variables you use.

Usage Notes for Memory Management

Operating systems: CMS, OS, VM/PC, and VSE

Under the operating systems listed, memory management in the SAS macro facility is controlled by the four SAS system options MSIZE=, MLEAVE=, MSYMSIZE=, and MWORK=.[7] In most cases you do not need to change the settings of the SAS options that affect macro facility memory management. However, if you write large or complex systems of macros you may increase the performance efficiency of the macro processor by changing the settings of the memory management options.

The memory management options can be specified only at SAS invocation. The only option for which an incorrect setting produces a message is the MWORK= option; the descriptions below can help you estimate the memory requirements of your macros.

MSIZE=12288
MSIZE=12K
: specifies the size of the area the macro processor allocates for itself at the beginning of a job or an interactive SAS session. Once allocated, this area cannot be changed during a job or session; however, the macro facility can acquire additional extents as needed. Since the additional memory may not be contiguous to the original allocation, SAS as a whole may use more memory and run slightly more slowly if additional extents are used. For maximum efficiency, choose a value for MSIZE= that does not require the macro facility to use additional extents.

  The MSIZE= area is divided into the MLEAVE= area (see the MLEAVE= option, below) and the area allowed for macro execution frames. A macro execution frame is an area of 2048 bytes that is available during macro execution. When a macro is compiled, the macro compiler packages the macro into units of 2048 bytes. If the compiled macro contains more units than execution frames are available, the macro processor shifts the units into or out of storage during macro execution, depending on which units are needed at a particular time. (The execution frames area cannot acquire additional extents.) Thus, you can

increase macro execution efficiency by setting the number of execution frames so that little shifting is needed. The number of frames available is determined by

FRAMES=(MSIZE-MLEAVE) / 2048 .

The macro facility requires at least one execution frame; the default values of MSIZE= and MLEAVE= allow for three frames. A small macro (see below) can be contained in one execution frame; a medium-sized macro requires two to three frames; and a large macro requires more than three frames.

The number of execution frames needed for a particular macro depends on the compiled length of the macro; the compiled length of a macro is approximately the length of its source statements. Therefore, one execution frame can contain about twenty-five 80-character lines of macro statements (or about fifty average lines since most statements occupy half a line or less). Macro comment statements, blank lines, and comments of the form /* *comment* */ do not count toward the total characters.

Since the macro facility requires at least one execution frame, the value of MSIZE= must be at least 2048 greater than the value of MLEAVE=. To increase the value of MSIZE=, use increments in multiples of 2048 (2K).

MLEAVE=6144
MLEAVE=6K

specifies the amount of the MSIZE= area that is to be reserved for symbol tables, work areas for the execution monitor for individual macros, and other items. The MLEAVE= area can acquire more extents if necessary; however, using additional extents may increase the amount of memory the SAS System as a whole uses. The amount of space required for MLEAVE= depends mainly on the number of execution monitor work areas needed (controlled by the number of macros executing at any point, including macros executing recursively) and the number and size of symbol tables needed for macro variables (see the MSYMSIZE= option, below). A very general formula for the value of the MLEAVE= option in production jobs is

MLEAVE=MSYMSIZE x 1/2 K x maximum recursion depth within job

To increase the value of MLEAVE=, use increments in multiples of 2048 (2K).

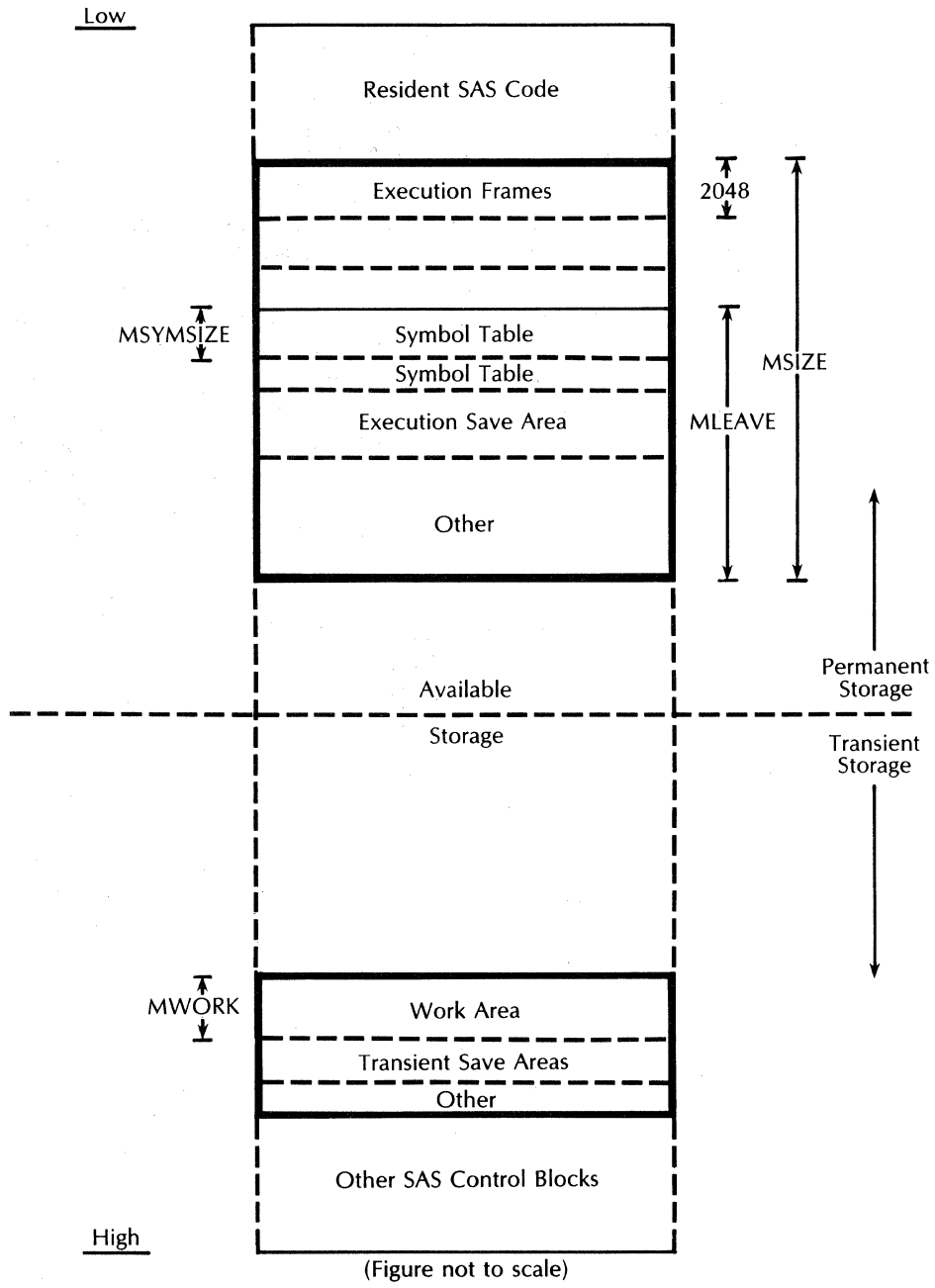


Figure 19.1 Macro Facility Memory Diagram

MSYMSIZE=1024
MSYMSIZE=1K

specifies the initial size of each macro language symbol table. (A symbol table stores all the macro variables and macro statement labels available to a particular referencing environment.) Since one macro variable with a value eight characters long requires about 60

bytes, the default value allows for about twenty macro variables with values of eight characters each. (The default value allows for about thirty-five macro statement labels if no macro variable values are present.) If you have more than twenty macro variables, thirty-five macro statement labels, or a combination of these in a referencing environment, or if the macro variables have long values, you may need to increase the size of the symbol tables. If the value of MSYMSIZE= is not sufficient for a given symbol table, the macro facility provides additional extents; however, using additional extents may increase the amount of memory the SAS System as a whole uses and may slow macro execution. Symbol tables are contained within the MLEAVE= area; therefore, if you increase the value of MSYMSIZE=, you may need to increase the values of MLEAVE= and MSIZE=. The value of MSYMSIZE= can be 1024 (1K) or a multiple of 1024.

MWORK=2048
MWORK=2K
: specifies the size of internal work areas for the macro facility. The maximum length of a macro variable value, a macro function argument, or a macro function result cannot exceed the value of the MWORK= option. If the length of one of these items is greater than the MWORK= value, the macro processor produces an error message stating that the work area size has been exceeded. (Since macro variable values, macro function arguments, and macro function results may contain one or more control characters in addition to their text, the maximum length for their text may be less than the value of MWORK=. The difference is usually less than 10 bytes.) The value of MWORK= may be 1024 (1K), 2048 (2K), 4096 (4K), or a multiple of 4096 up to 28672 (28K).

EXAMPLES

Substitution of Macro Parameters: Example 1

This example illustrates different SAS statements produced by substituting different values for macro parameters. The macro allows you to specify a list of variables to be printed by the PRINT procedure and a list of variables for which the procedure prints a total. (See the PRINT procedure for details.) The variables to appear in the VAR statement are given in parameter VAR; those to appear in the SUM statement are given in parameter SUM.

The macro definition:

```
%MACRO P(VAR,SUM);
   PROC PRINT;
      VAR &VAR;
      SUM &SUM;
   RUN;
   %MEND P;
```

The call

```
%P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

causes SAS to see the statements in **Output 19.33**.

Output 19.33 Macro Invocation Supplying Values for Two Parameters

```
13        + PROC PRINT;                                                    1-%p
14        + VAR SCHOOL DISTRICT ENROLLMT;                                  1-%p
15        + SUM ENROLLMT;                                                  1-%p
16        + RUN;                                                           1-%p
```

This execution of the PRINT procedure prints values for variables SCHOOL, DISTRICT, and ENROLLMT with a sum for ENROLLMT. The call

```
%P(DISTRICT ENROLLMT)
```

causes SAS to receive the statements in **Output 19.34**.

Output 19.34 Macro Invocation Supplying a Value for One Parameter

```
22        + PROC PRINT;                                                    1-%p
23        + VAR DISTRICT ENROLLMT;                                         1-%p
24        + SUM;                                                           1-%p
25        + RUN;                                                           1-%p
```

Here the PRINT procedure prints values for ENROLLMT and DISTRICT without any sums. Since the invocation does not contain a comma, the entire character string in parentheses in the call is the value of the first macro parameter. In this PROC PRINT step the SUM statement does not contain a variable list; however, a SUM statement without variables is an acceptable statement in the PRINT procedure. Macro P takes advantage of this fact to allow you to choose whether to specify sum variables when you call the macro.

Macro Printing Options: Example 2

This example uses macro P above to illustrate information about macro execution that appears on the log when various SAS options are in effect. All calls in this example were made in a job run under OS batch. For clarity, the OPTIONS statement specifies the setting of all macro printing options in this example, regardless of their default values.

Using the NOMPRINT, NOMACROGEN, NOSYMBOLGEN, and NOMLOGIC options These statements:

```
OPTIONS NOMPRINT NOMACROGEN NOSYMBOLGEN NOMLOGIC;
%P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

produce the message on the SAS log shown in **Output 19.35**.

Output 19.35 Text Displayed Using NOMPRINT, NOMACROGEN, NOSYMBOLGEN, and NOMLOGIC

```
29          OPTIONS NOMPRINT NOMACROGEN NOSYMBOLGEN NOMLOGIC;
NOTE: THE PROCEDURE PRINT USED 0.29 SECONDS AND 464K AND PRINTED PAGE 3.

30          %P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

No information about the execution of the macro appears, although a message indicating completion of the procedure is printed.

Using the MPRINT option The statements

```
OPTIONS MPRINT;
%P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

place the information shown in **Output 19.36** into the SAS log.

Output 19.36 Text Displayed Using MPRINT

```
46          OPTIONS MPRINT;
47          %P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
48        + PROC PRINT;                                                  1-%p
49        + VAR SCHOOL DISTRICT ENROLLMT;                                1-%p
50        + SUM ENROLLMT;                                                1-%p
51        + RUN;                                                         1-%p
```

The statements produced by the macro appear in the same form that SAS sees them. For ease in reading, each statement begins on a separate line, and one space is printed between words. The plus signs (+) distinguish the SAS statements generated by the macro from statements not generated by the macro. The name of the macro that generated each line and the level of nesting (in this case, one level) also appear. The MPRINT option suppresses the MACROGEN, SYMBOLGEN, and MLOGIC options during macro execution.

Using the MACROGEN option The following statements:

```
OPTIONS NOMPRINT MACROGEN NOSYMBOLGEN NOMLOGIC;
%P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

produce the information shown in **Output 19.37** on the SAS log.

Output 19.37 Text Displayed Using MACROGEN

```
57          OPTIONS NOMPRINT MACROGEN NOSYMBOLGEN NOMLOGIC;
58          %P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
60        +PROC PRINT;                                                   1-%p
62        +VAR &VAR;                                                     1-%p
65        +SUM &SUM;                                                     1-%p
68        +RUN;                                                          1-%p
```

The MACROGEN option prints the statements generated by the macro but does not show the substitution of macro variable values into the macro variable references. Thus, the terms &VAR and &SUM appear in the generated statements. The NOMPRINT option must be in effect to use the MACROGEN option with statements produced by a macro.

Using the MACROGEN and SYMBOLGEN options The following statements:

```
OPTIONS NOMPRINT MACROGEN SYMBOLGEN NOMLOGIC;
%P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
```

print the information in **Output 19.38** on the SAS log.

Output 19.38 Text Displayed Using MACROGEN and SYMBOLGEN

```
72          OPTIONS NOMPRINT MACROGEN SYMBOLGEN NOMLOGIC;
73          %P(SCHOOL DISTRICT ENROLLMT, ENROLLMT)
75       +PROC PRINT;                                                    1-%p
77       +VAR &VAR                                                       1-%p
78       + SCHOOL DISTRICT ENROLLMT                                      2-&SYMBOL
77       +       ;                                                       1-%p
80       +SUM &SUM                                                       1-%p
81       + ENROLLMT                                                      2-&SYMBOL
80       +       ;                                                       1-%p
83       +RUN;                                                           1-%p
```

The SYMBOLGEN option produces a listing of the values for macro variables (also called symbolic variables) that are being substituted into the text produced. The SYMBOLGEN lines are separate from the MACROGEN lines. The NOMPRINT option must be in effect to use the SYMBOLGEN option to show macro variable resolution within a macro.

Using the MLOGIC option The following macro, P2, contains a choice of actions within the macro to illustrate the MLOGIC option. Macro P2 allows you to choose whether an explanatory line is printed as a title or a footnote:

```
%MACRO P2(VAR,SUM,WHERE);
   PROC PRINT;
      VAR &VAR;
      SUM &SUM;
   %IF %UPCASE(&WHERE)=TITLE %THEN %DO;
      TITLE "REPORT FOR 1984-85 SCHOOL YEAR";
      FOOTNOTE;
      %END;
   %ELSE %DO;
      TITLE;
      FOOTNOTE "REPORT FOR 1984-85 SCHOOL YEAR";
      %END;
   RUN;
   %MEND P2;
```

When the uppercase equivalent of the value of macro WHERE is TITLE, the line REPORT FOR 1984-85 SCHOOL YEAR appears in a TITLE statement; otherwise the line appears as a footnote.

The following call:

```
OPTIONS NOMPRINT NOMACROGEN NOSYMBOLGEN MLOGIC;
%P2(SCHOOL DISTRICT ENROLLMT, ENROLLMT, TITLE)
```

appears on the SAS log as shown in **Output 19.39**.

Output 19.39 Text Displayed Using MLOGIC

```
104          OPTIONS NOMPRINT NOMACROGEN NOSYMBOLGEN MLOGIC;
88        >%UPCASE(&WHERE)=TITLE                                          1-<MLOGIC>
105          %P2(SCHOOL DISTRICT ENROLLMT, ENROLLMT, TITLE)
```

The generated line

```
>%UPCASE(WHERE)=TITLE
```

is produced by the MLOGIC option and indicates the point at which the %IF expression is evaluated. Since the uppercase value of WHERE was TITLE, macro execution followed the path for a true condition in the %IF statement. Note that the NOMPRINT option must be in effect to use the MLOGIC option, and line numbers produced by the MLOGIC option are not in sequence with other line numbers.

Using the MLOGIC, MACROGEN, and SYMBOLGEN options The following macro call:

```
OPTIONS NOMPRINT MACROGEN SYMBOLGEN MLOGIC;
%P2(SCHOOL DISTRICT ENROLLMT, ENROLLMT, TITLE)
```

places the information shown in **Output 19.40** into the SAS log.

Output 19.40 Text Displayed Using MLOGIC, MACROGEN, and SYMBOLGEN

```
125          OPTIONS NOMPRINT MACROGEN SYMBOLGEN MLOGIC;
126          %P2(SCHOOL DISTRICT ENROLLMT, ENROLLMT, TITLE)
128       +PROC PRINT;                                                    1-%p2
130       +VAR &VAR                                                       1-%p2
131       + SCHOOL DISTRICT ENROLLMT                                      2-&SYMBOL
130       +       ;                                                       1-%p2
133       +SUM &SUM                                                       1-%p2
134       + ENROLLMT                                                      2-&SYMBOL
133       +       ;                                                       1-%p2
107       >TITLE                                                          2-&SYMBOL
108       >TITLE                                                          2-%UPCASE
106       >%UPCASE(&WHERE)=TITLE                                          1-<MLOGIC>
136       +TITLE "REPORT FOR 1984-85 SCHOOL YEAR";                        1-%p2
138       +FOOTNOTE;                                                      1-%p2
140       +RUN;                                                           1-%p2
```

This combination of options provides the most detailed information possible about the execution of a macro, but the output from even a very simple macro such as P2 is bulky. In general, we recommend that you use the MPRINT option to see the SAS statements produced by a macro; use the MLOGIC and related options only when you need the level of detail they provide.

Repeated Invocations of a Procedure: Example 3

This example illustrates using a %DO %WHILE loop to invoke a procedure repeatedly with different option values. In this case the value of the H= option

in the GSLIDE procedure changes in each invocation of the procedure. The example also uses two automatic macro variables, SYSDAY and SYSDATE.

Macro HIGH allows you to see the results of PROC GSLIDE with several different character heights for the two title lines. The FOOTNOTE statement identifies the value of the H= option in each procedure invocation. Macro HIGH simplifies selecting a height value for the TITLE statements since you can compare the different results quickly, without retyping statements, and note the best height for further use. GSLIDE is a SAS/GRAPH procedure.

```
%MACRO HIGH(HEIGHT,LIMIT);
   %DO I=&HEIGHT %TO &LIMIT;
      PROC GSLIDE;
         TITLE1;
         TITLE2 H=&HEIGHT F=COMPLEX "&SYSDAY";
         TITLE4 H=&HEIGHT F=COMPLEX "&SYSDATE";
         FOOTNOTE "HEIGHT IS &HEIGHT";
      RUN;
      %END;
   %MEND HIGH;
%HIGH(1,5)
```

For example, suppose you call macro HIGH on Friday, July 26, 1985. The %DO loop is executed five times, and the first execution produces the SAS statements shown in **Output 19.41.**

Output 19.41 First PROC GSLIDE Step Generated by Macro

```
81      + PROC GSLIDE;                                                  1-%high
82      + TITLE1;                                                       1-%high
83      + TITLE2 H=1 F=COMPLEX "FRIDAY";                                1-%high
84      + TITLE4 H=1 F=COMPLEX "26JUL85";                               1-%high
85      + FOOTNOTE "HEIGHT IS 1";                                       1-%high
86      + RUN;                                                          1-%high
```

The GSLIDE procedure displays the lines FRIDAY and 26JUL85 using the COMPLEX font in characters 1 line high. Each execution of the %DO loop increases the value of HEIGHT by 1. The final set of statements produced is shown in **Output 19.42.**

Output 19.42 Final PROC GSLIDE Step Generated by Macro

```
105     + PROC GSLIDE;                                                  1-%high
106     + TITLE1;                                                       1-%high
107     + TITLE2 H=5 F=COMPLEX "FRIDAY";                                1-%high
108     + TITLE4 H=5 F=COMPLEX "26JUL85";                               1-%high
109     + FOOTNOTE "HEIGHT IS 5";                                       1-%high
110     + RUN;                                                          1-%high
```

When the value of HEIGHT is incremented to 6, the expression in the %DO %WHILE statement is no longer true and the %DO loop ceases to be executed.

Producing Parts of SAS Statements: Example 4

This example illustrates analyzing a planned section of SAS code for its repetitive features and producing the repetitive code through parts of SAS statements instead of complete SAS statements.

Suppose you have a survey of persons ages 30-59 whose ages in years are recorded in variable AGEYR. You want to create a new variable AGE3 in which persons are assigned to a three-year age group: 30-32, 33-35, 36-38, and so on. IF-THEN statements are a flexible way to recode variables, since you can recode the variables by any interval, but a long series of IF-THEN/ELSE statements requires a lot of coding.[8] The DATA step you plan in this example is

```
DATA AGES;
   INFILE IN;
   INPUT ID REGION AGEYR;
   IF 30<=AGEYR<33 THEN AGE3=30;
   ELSE IF 33<=AGEYR<36 THEN AGE3=33;
   ELSE IF 36<=AGEYR<39 THEN AGE3=36;
   ELSE IF 39<=AGEYR<42 THEN AGE3=39;
   ELSE IF 42<=AGEYR<45 THEN AGE3=42;
   ELSE IF 45<=AGEYR<48 THEN AGE3=45;
   ELSE IF 48<=AGEYR<51 THEN AGE3=48;
   ELSE IF 51<=AGEYR<54 THEN AGE3=51;
   ELSE IF 54<=AGEYR<57 THEN AGE3=54;
   ELSE IF 57<=AGEYR THEN AGE3=57;
```

Macro CODEIT simplifies coding this DATA step and can be adapted to many situations. It generates a series of IF-THEN/ELSE statements to recode the DATA step variable identified in macro variable OLD into the DATA step variable identified in NEW. The macro definition is

```
%MACRO CODEIT(START=, STOP=, COUNT=, OLD=, NEW=);
   IF &START
   %DO X=&START+&COUNT %TO &STOP %BY &COUNT;
      <=&OLD<&X THEN &NEW=&START;
      %LET START=&X;
      ELSE IF &START
      %END;
   <=&OLD THEN &NEW=&START;
   %MEND CODEIT;
```

The DATA step above then becomes

```
DATA AGES;
   INFILE IN;
   INPUT ID REGION AGEYR;
%CODEIT(START=30,STOP=59,COUNT=3,OLD=AGEYR,NEW=AGE3)
```

In this example the repetitive pattern in the DATA step begins with the symbols <= in the first IF statement and ends with the ELSE IF *value* code in the next statement. Thus, you can enclose this code in an iterative %DO loop for the number of repetitions you need. The portions of SAS statements before and after the repetitive pattern are produced outside the loop.

Macro CODEIT produces SAS statements using this logic:

1. In the first section of constant text the macro variable reference &START is replaced by the value of START and the first part of a SAS statement is produced:

```
IF 30
```

2. The %DO loop begins execution. The first value of X is determined by adding the interval COUNT to the value of START. In this case, the first value of X is 33. Notice that &START still has the value that begins the lowest grouping. The macro variable references in the constant text are resolved and the next piece of constant text is produced:

```
<=AGEYR<33 THEN AGE3=30;
```

   This complete SAS statement has now been produced:

```
IF 30<=AGEYR<33 THEN AGE3=30;
```

   The %LET statement changes the value of START to the current value of X (33). This action means that, after the value of X is incremented for the next execution of the %DO loop, START will contain the former value of X. The next portion of constant text is produced:

```
ELSE IF 33
```

   At this point the first execution of the %DO loop is complete. The value of X is now incremented by the value of COUNT and the second execution of the loop begins. The first section of text produced is

```
<=AGEYR<36 THEN AGE3=33;
```

   which completes the SAS statement begun in the previous execution of the loop. The second complete SAS statement produced is thus

```
ELSE IF 33<=AGEYR<36 THEN AGE3=33;
```

   The %LET statement changes the value of START to the current value of X (36), and the next portion of constant text is produced:

```
ELSE IF 36
```

3. At the end of the last execution of the %DO loop, this partial SAS statement has been produced:

```
ELSE IF 57
```

   The value of X is now incremented to 60 and the %DO loop is no longer executed. Then the last section of constant text is produced:

```
<=AGEYR THEN AGE3=57;
```

   The final SAS statement is therefore

```
ELSE IF 57<=AGEYR THEN AGE3=57;
```

See **Output 19.43.**

Output 19.43 Repetitive DATA Step Statements Generated by Macro

```
212          DATA AGES;
213             INFILE IN;
214             INPUT ID REGION AGEYR;
215          %CODEIT(START=30,STOP=59,COUNT=3,OLD=AGEYR,NEW=AGE3)
216       + IF 30 <=AGEYR<33 THEN AGE3=30;                                  1-%codeit
217       + ELSE IF 33 <=AGEYR<36 THEN AGE3=33;                             1-%codeit
218       + ELSE IF 36 <=AGEYR<39 THEN AGE3=36;                             1-%codeit
219       + ELSE IF 39 <=AGEYR<42 THEN AGE3=39;                             1-%codeit
220       + ELSE IF 42 <=AGEYR<45 THEN AGE3=42;                             1-%codeit
221       + ELSE IF 45 <=AGEYR<48 THEN AGE3=45;                             1-%codeit
222       + ELSE IF 48 <=AGEYR<51 THEN AGE3=48;                             1-%codeit
223       + ELSE IF 51 <=AGEYR<54 THEN AGE3=51;                             1-%codeit
224       + ELSE IF 54 <=AGEYR<57 THEN AGE3=54;                             1-%codeit
225       + ELSE IF 57 <=AGEYR THEN AGE3=57;                                1-%codeit
```

Transferring Constant Values with the SYMPUT Function and Macro Variable References: Example 5

This example uses values produced by PROC MEANS in creating option values to be used with PROC PLOT. The program creates a plot of SALARY*AGE for each BY group in data set EMPLOY with a reference line drawn on the horizontal axis of each plot at the mean salary for that group. The process requires three stages:

1. The MEANS procedure produces values and stores them in a SAS data set.
2. A DATA _NULL_ step reads that data set, modifies the values as necessary, and uses the SYMPUT function to create macro variables containing those values.
3. The PROC PLOT steps use macro variable references to retrieve the appropriate macro variable values.

This program is designed for large data sets, and using the SYMPUT function considerably increases the program's efficiency. Since the SYMPUT function allows you to execute the program as one job instead of two (one to produce the MEANS output and one in which you code the values from MEANS into the PROC PLOT steps), the SAS and operating system overhead is reduced.

```
DATA EMPLOY;
   INPUT SEX $ AGE SALARY;
   CARDS;
F 21 17000
F 18 10500
M 19 10900
M 33 31000
F 32 27000
M 23 21000
M 24 21000
PROC SORT;
   BY SEX;
PROC MEANS;
   BY SEX;
   VAR SALARY;
   OUTPUT OUT=MEAN N=TOT MEAN=MSAL;
PROC PRINT;
RUN;
TITLE "AGE * SALARY";
DATA _NULL_;
   SET MEAN END=EOF;
```

```
   LENGTH NCNT $3;
   RETAIN FIRSTOBS 1;
   OBS+TOT;
   NCNT=LEFT(_N_);
   CALL SYMPUT('FIRST'||NCNT,FIRSTOBS);
   CALL SYMPUT('LAST'||NCNT,OBS);
   CALL SYMPUT('REF'||NCNT,MSAL);
   FIRSTOBS+OBS;
   IF EOF THEN CALL SYMPUT('TOTAL',_N_);
   RUN;
%MACRO PLOTIT;
   %DO I=1 %TO &TOTAL;
      PROC PLOT DATA=EMPLOY(FIRSTOBS=&&FIRST&I OBS=&&LAST&I);
         PLOT AGE*SALARY / HREF=&&REF&I;
      TITLE2 "GROUP&I";
      %END;
   RUN;
   %MEND;
%PLOTIT
```

DATA step EMPLOY creates the data set to be analyzed. The SORT procedure sorts EMPLOY by variable SEX, and the MEANS procedure calculates the number of observations in each BY group (stored in variable TOT) and the mean salary for each BY group (stored in variable MSAL). The output of the MEANS procedure is contained in the data set MEAN, which has one observation for each BY-group. See **Output 19.44**.

Output 19.44 PROC MEANS

VARIABLE	N	MEAN	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	STD ERROR OF MEAN	SUM	VARIANCE	C.V.
SEX=F									
SALARY	3	18166.66667	8311.638427	10500.00000	27000.00000	4798.726683	54500.00000	69083333.33	45.752
SEX=M									
SALARY	4	20975.00000	8205.841415	10900.00000	31000.00000	4102.920708	83900.00000	67335833.33	39.122

The next DATA step uses information from data set MEAN to create several macro variables for macro PLOTIT. Since you are using this DATA step only to create macro variables, you can code DATA _NULL_ to use program statements without creating a SAS data set.

In the first execution of the DATA step:

1. the value of OBS is the number of the last observation in the first BY group of EMPLOY, since TOT is the number of observations in each BY group.
2. _N_ is the number of the BY group being processed.
3. NCNT is a character variable created by left-aligning the value of _N_. It is created so that the CALL SYMPUT statements following it can create the name of a macro variable by concatenating the first part of the name with the character value of NCNT.
4. the first CALL SYMPUT produces a macro variable FIRST1 with a value of 1, since FIRSTOBS was initialized to 1 with the RETAIN statement.

5. the second CALL SYMPUT statement produces a macro variable LAST1 with a value of the number of observations in the first BY-group (OBS).
6. the third CALL SYMPUT statement produces a macro variable REF1 with a value of the mean salary for persons in BY-group 1.
7. after the three macro variables are created, FIRSTOBS is increased by the number of observations in the first BY group (OBS). Thus, FIRSTOBS now contains the number of the observation in EMPLOY that begins the second BY group.

In the second (and last) execution:

1. OBS is equal to the number of the last observation in the last BY group in EMPLOY, that is, the total number of observations in EMPLOY.
2. _N_ and NCNT are both 2 (the number of the last BY group in EMPLOY).
3. macro variable FIRST2 has a value representing the number of the first observation in the second BY group.
4. the value of macro variable LAST2 is the number of the last observation in the second BY group, that is, the last observation in data set EMPLOY.
5. macro variable REF2 contains the mean salary for persons in BY group 2.
6. FIRSTOBS is equal to the number of the first observation in the (nonexistent) third BY group.
7. since EOF has a value of 1 (true) during the second execution of FORPLOTS, the IF statement creates a macro variable named TOTAL whose value represents the number of BY groups in EMPLOY (that is, _N_).

Thus, this job creates seven macro variables: FIRST1, LAST1, REF1, FIRST2, LAST2, REF2, and TOTAL.

The %DO loop in macro PLOTIT is executed as many times as the value of macro variable TOTAL indicates, that is, once for each BY group. For each plot, the observations to read from EMPLOY are selected automatically. The indirect macro variable references &&FIRST&I and &&LAST&I resolve to the value of FIRST1 and the value of LAST1. Thus, the first time the %DO loop is executed the FIRSTOBS= option has a value of 1, since that is the value of macro variable FIRST1. The OBS= option has the number of the last observation to read from EMPLOY for the first BY group, since that is the value of LAST1. A line is drawn at the mean salary of group 1 (the value of REF1). The second (and last) time the %DO loop is executed, the FIRSTOBS= value is the number of the first observation in the second BY group, and the OBS= value is the number of the last observation in the file. A line appears on the plot at the mean salary for BY group 2 (given in macro variable REF2).

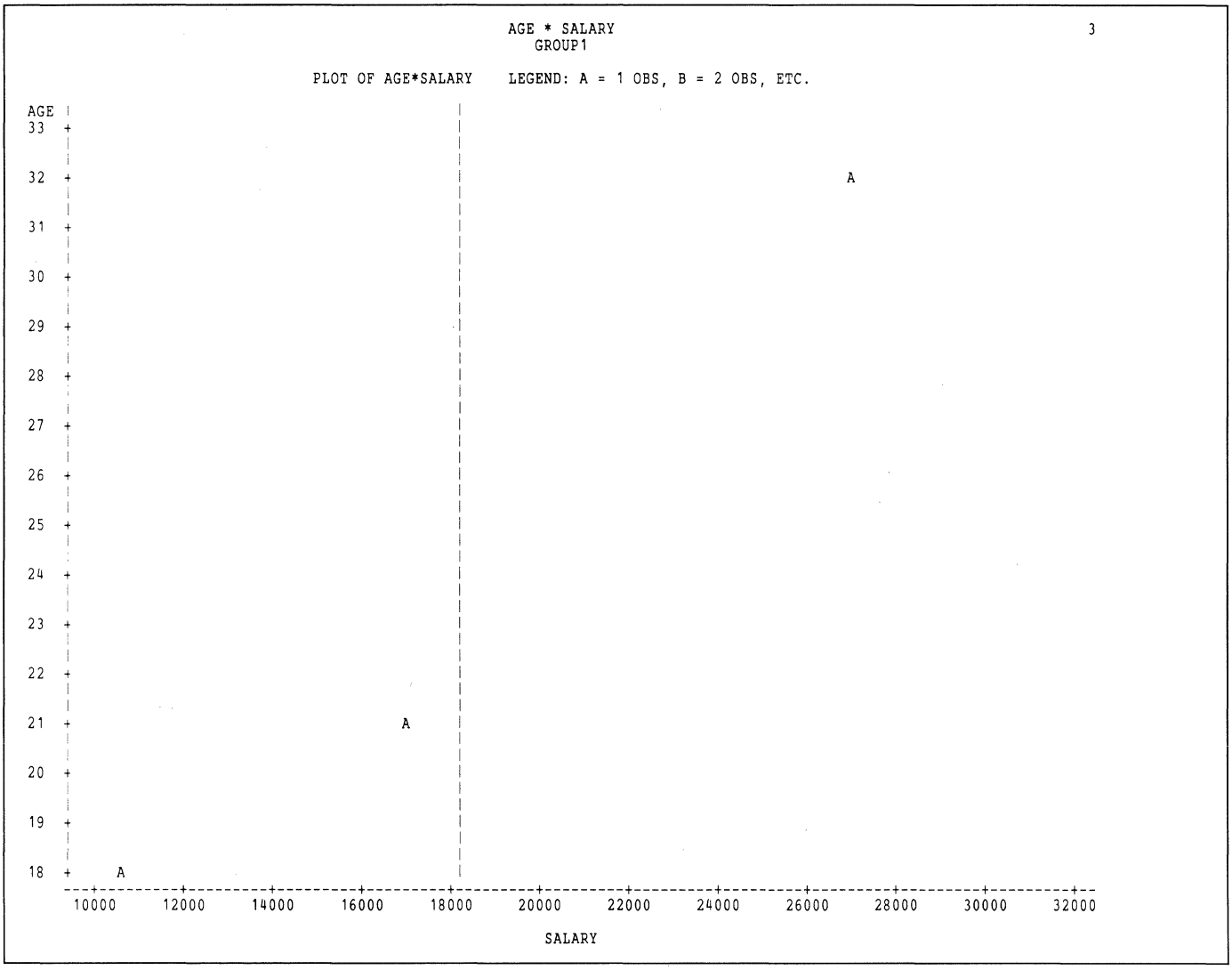
The SAS statements generated by macro PLOTIT are shown in **Output 19.45**.

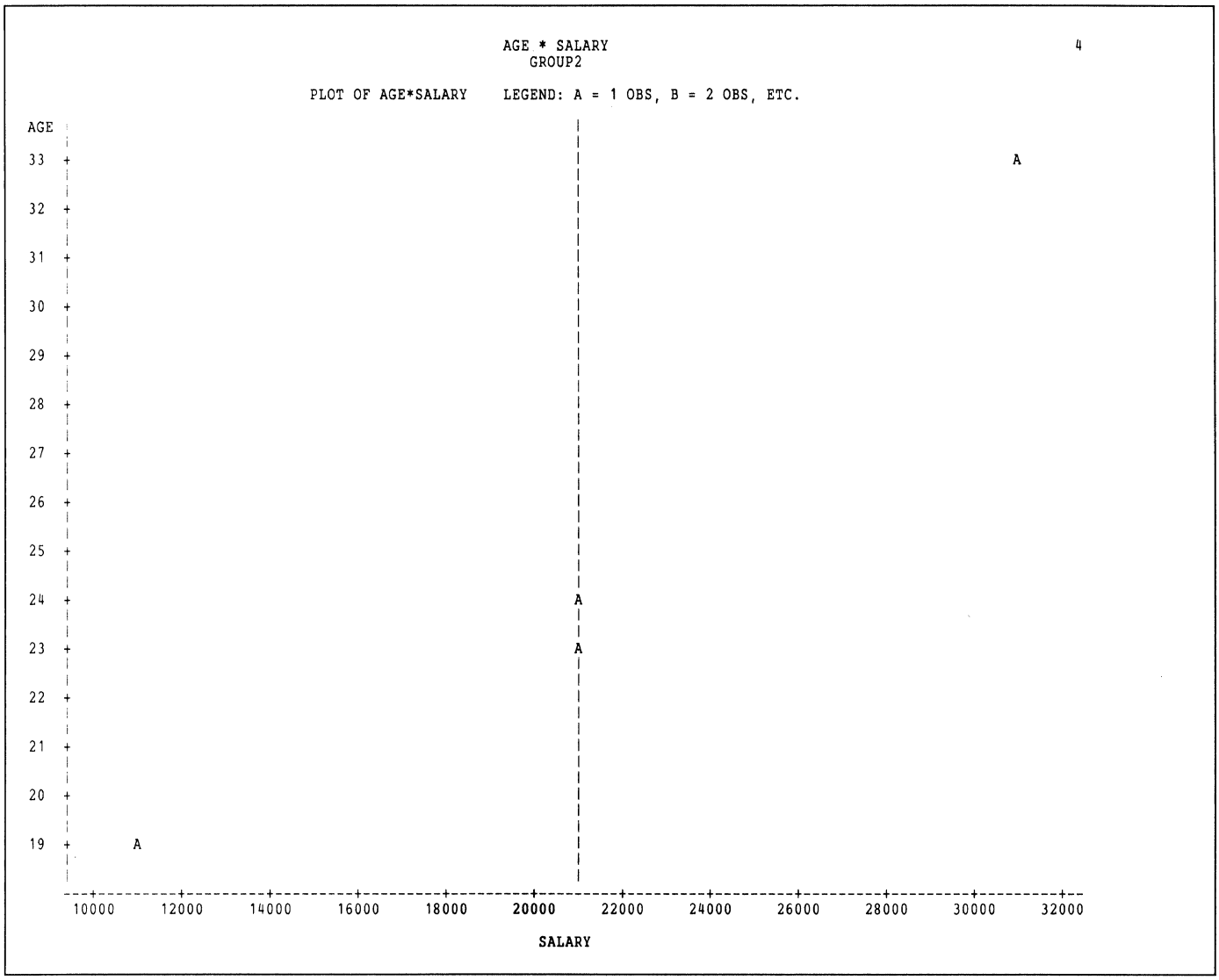
Output 19.45 PROC PLOT Steps Generated by Macro

```
278       + PROC PLOT DATA=                                                    1-%plotit
279       +EMPLOY(FIRSTOBS= 1 OBS= 3);                                         1-%plotit
280       + PLOT AGE*SALARY / HREF= 18166.67;                                  1-%plotit
281       + TITLE2 "GROUP1";                                                   1-%plotit
282       + PROC PLOT DATA                                                     1-%plotit
284       +EMPLOY(FIRSTOBS= 4 OBS= 7);                                         1-%plotit
285       + PLOT AGE*SALARY / HREF= 20975;                                     1-%plotit
286       + TITLE2 "GROUP2";                                                   1-%plotit
287       + RUN;                                                               1-%plotit
```

The plots are shown in **Output 19.46**.

Output 19.46 Plot Generated for Each Group





Transferring Variable Information with the SYMPUT and SYMGET Functions: Example 6

This example uses the SYMPUT and SYMGET functions to transfer information from a data set created by PROC SUMMARY into a data set containing salary information for employees of Dusty Department Store. The new data set contains each employee's name, department, salary, the employee's salary as a percentage of his department's total salary expenditure and of store's total salary expenditure, and the mean salary for his department.

```
DATA DUSTY;
   INPUT DEPT $ NAME $ SALARY @@;
   CARDS;
BEDDING WATLEE 18000      BEDDING IVES 16000
BEDDING PARKER 9000       BEDDING GEORGE 8000
BEDDING JOINER 8000       CARPET KARO 20000
CARPET RAY 12000          CARPET JONES 9000
CARPET MCNAIR 10000       CARPET THOMAS 12000
GIFTS JOHNSON 8000        GIFTS MATTHEW 19000
```

```
GIFTS RANKIN 7000          KITCHEN BANKS 14000
KITCHEN WHITE 8000         KITCHEN CANNON 15000
KITCHEN MARKS 9000         TV SMITH 8000
TV JONES 9000              TV MORSE 16000
TV ROGERS 15000
;
PROC SUMMARY;
   CLASS DEPT;
   VAR SALARY;
   OUTPUT OUT=STATS N=N_SAL MEAN=M_SAL SUM=S_SAL;
PROC PRINT;
TITLE "SUMMARY OF SALARY INFORMATION";
TITLE2 "FOR DUSTY DEPARTMENT STORE";
DATA _NULL_;
   SET STATS;
   IF _N_ =1 THEN DO;
      CALL SYMPUT ('S_TOT',S_SAL);
      CALL SYMPUT ('M_TOT',M_SAL);
      END;
   ELSE DO;
      CALL SYMPUT('M'||DEPT,M_SAL);
      CALL SYMPUT('S'||DEPT,S_SAL);
      CALL SYMPUT('N'||DEPT,N_SAL);
      END;
DATA NEW;
   SET DUSTY;
   PCTDEPT=(SALARY / SYMGET('S'||DEPT)) * 100;
   DEPTMEAN=SYMGET('M'||DEPT);
   NUM=SYMGET('N'||DEPT);
   PCTTOT=(SALARY / SYMGET('S_TOT')) * 100;
PROC PRINT SPLIT="*";
   LABEL PCTDEPT="PERCENT OF *DEPARTMENT"
         PCTTOT="PERCENT OF *   TOTAL"
         DEPTMEAN="MEAN SALARY OF *DEPARTMENT"
         NUM="NUMBER OF EMPLOYEES*   IN DEPARTMENT";
TITLE "SALARY PROFILES FOR EMPLOYEES";
TITLE2 "OF DUSTY DEPARTMENT STORE";
RUN;
```

The DATA DUSTY step creates a SAS data set containing each employee's department, name, and salary. The SUMMARY procedure analyzes data set DUSTY to produce an output data named STATS containing the number of observations (variable N_SAL), mean salary (variable M_SAL), and sum of salaries (variable S_SAL), for each department and for the store as a whole as well as the special variables _TYPE_ and _FREQ_.

The DATA _NULL_ step creates macro variables from variables in STATS:

- Since the first observation in STATS contains information about all observations in DUSTY (_TYPE_=0), the first execution of the DATA step places the sum of salaries for all employees (S_SAL) into the value of macro variable S_TOT. The mean salary for all employees (M_SAL) is placed into the value of macro variable M_TOT.
- In the second execution of the DATA _NULL_ step, a macro variable is created by concatenating the letter M and the name of a department, in this case BEDDING. MBEDDING contains the mean salary (from M_SAL) for employees of the bedding department. Likewise, macro variable

SBEDDING contains the sum of the salaries of employees in that department, and NBEDDING contains the number of employees in that department.

- Subsequent executions of the DATA step produce analogous macro variables for other departments.

In all, the DATA _NULL_ step produces 17 macro variables: three for each of the five departments, plus two for the store as a whole.

Data set NEW contains each employee's salary as a percentage of the departmental salary expenditure (PCTDEPT) and as a percentage of the store's total salary expenditure (PCTTOT). It also contains the mean salary of that employee's department (DEPTMEAN) and the number of employees in the department (NUM) for each observation. In the expression that calculates PCTDEPT, the SYMGET function retrieves the value of the macro variable whose name is obtained by concatenating the letter S and the value of the DEPT variable. DEPTMEAN and NUM are constructed in the same way. PCTTOT is created by dividing the employee's salary by the total of salaries in the store (returned by SYMGET('S_TOT')).

Finally, the PRINT procedure lists the resulting information for each employee: department, name, salary, salary as a percentage of the department's total salaries, salary as a percent of the store's total salaries, mean salary of the employee's department, and number of employees in the department.

The difference between retrieving a macro variable value with the SYMGET function as in this example and with a macro variable reference as in the previous example is that the SYMGET function retrieves values during DATA step execution, while a macro variable reference retrieves them during compilation. That is, information retrieved by a macro variable reference is constant during DATA or PROC step execution; the SYMGET function can retrieve different information in each execution of the DATA step. In this example the SYMGET function uses the values of DATA step variables (known only during the execution of the DATA step for a particular observation) to determine which macro variable value to retrieve.

A Simple Interactive System of Macros: Example 7

This example illustrates a simple system of macros in which the macro you invoke calls other macros to perform various tasks. (A macro that invokes other macros in a system is sometimes called a *driver macro*.) The example also illustrates several possible programming techniques for the macro language. The driver macro, REPORT, is listed first; the other macros follow it in alphabetical order.

In this example, you can select either a chart or a printed listing of the monthly sales to date for each sales representative from a SAS data set that your company's information center creates each day. The data set containing the information is shown in **Output 19.47**.

```
PROC PRINT DATA=INPUT.CURRENT;
   FORMAT MONTH DATE7.;
```

Output 19.47 Data Set Containing Information for Reports

OBS	MONTH	SALESREP	SALES
1	01APR85	Herbert	$12,500
2	01APR85	Lynn	$15,000
3	01APR85	Reel	$11,000
4	01APR85	Tyndall	$13,500
5	01APR85	Douglas	$15,000
6	01APR85	Cannon	$14,000

The macros composing the system are

```
*OPERATING SYSTEMS: CMS, OS, AND VM/PC;
OPTIONS DQUOTE CENTER NOMPRINT NOMACROGEN NOSYMBOLGEN NOMLOGIC;

%MACRO REPORT;
   %GLOBAL BAD DATE NUM CHOICE TIME EOF;
   %IF &SYSENV=BACK %THEN %DO;
      %SPACES(3)
      %PUT THIS EXAMPLE DOES NOT WORK IN BATCH MODE.;
      %SPACES(3)
      %GO TO STOP;
      %END;
   %MENU
   %IF &SYSBUFFR ¬= %THEN %BUFFER;
   %IF &SYSSCP=OS %THEN %DO;
      %TSO ALLOC F(INPUT) DA('SALES.DATA.LIBRARY') SHR;
      %RC
      %END;
   %ELSE %IF &SYSSCP=CMS %THEN %DO;
      %CMS FILEDEF INPUT DISK D D A;
      %RC
      %END;
   %ELSE %DO;
      %PUT THIS EXAMPLE IS AVAILABLE;
      %PUT %STR(     ONLY UNDER INTERACTIVE CMS AND TSO.);
      %LET BAD=1;
      %END;
   %IF &BAD %THEN %GOTO STOP;

   %IF &CHOICE=1 %THEN %DO;
      %CHECK(CURRENT)
      %IF &EOF=E %THEN %DO;
         %MESSAGE
         %GO TO STOP;
         %END;
      PROC CHART DATA=INPUT.CURRENT;
         VBAR SALESREP / SUMVAR=SALES;
         TITLE1 "SALES CHART TO DATE BEGINNING &DATE";
         TITLE2 "&NUM SELLING DAYS";
      RUN;
      %END;

   %ELSE %IF &CHOICE=2 %THEN %DO;
      %CHECK(CURRENT)
      %IF &EOF=E %THEN %DO;
         %MESSAGE
         %GO TO STOP;
         %END;
     PROC PRINT DATA=INPUT.CURRENT;
        VAR SALES;
        ID SALESREP;
        TITLE1 "MONTHLY SALES TO DATE--REPORT BEGINNING &DATE";
        TITLE2 "&NUM SELLING DAYS";
     RUN;
     %END;
```

```
   %ELSE %DO;
      %ERROR
      %IF &BAD AND &TIME %THEN %GO TO STOP;
      %END;

   %SPACES(2)
   %PUT TO REQUEST ANOTHER REPORT, ENTER %NRSTR(%REPORT);
   %STOP: %LET BAD=0;
   %LET TIME=0;
   %MEND REPORT;

 /*****************************************************************/

%MACRO BUFFER;
      %PUT EXTRA INFORMATION IGNORED:;
      %PUT &SYSBUFFR;
      %PUT EXECUTION WILL CONTINUE.;
      %PUT PLEASE BE CAREFUL TO FOLLOW THE INSTRUCTIONS.;
   %MEND BUFFER;

 /*****************************************************************/

%MACRO CHECK(SALES);
   DATA _NULL_;
      I=1;
      IF NNN=0 THEN DO;
         CALL SYMPUT('EOF','E');
         STOP;
         END;
      ELSE DO;
         SET INPUT.&SALES POINT=I NOBS=NNN;
         M=MONTH;
         NOW=INPUT("&SYSDATE",DATE7.);
         DO WHILE(M<NOW);
            W=WEEKDAY(M);
            IF 2<=W<=6 THEN SELLDAYS+1;
            M=M+1;
            END;
         CALL SYMPUT('NUM',TRIM(PUT(SELLDAYS,WORDS12.)));
         CALL SYMPUT('EOF','F');
         CALL SYMPUT('DATE',PUT(MONTH,DATE7.));
         STOP;
         END;
   RUN;
   %MEND CHECK;

 /*****************************************************************/

%MACRO ERROR;
   %SPACES(3)
   %IF &TIME=1 %THEN %DO;
      %PUT YOU HAVE EXCEEDED THE MAXIMUM NUMBER OF ATTEMPTS;
      %PUT %STR(   FOR THIS PROCESS.);
      %PUT EXECUTION WILL CEASE.;
```

```
      %PUT PLEASE READ YOUR INSTRUCTION SHEET OR CONSULT YOUR;
      %PUT %STR(   SUPERVISOR IF YOU NEED ASSISTANCE.);
      %LET BAD=1;
      %END;
   %ELSE %DO;
      %LET BAD=0;
      %LET TIME=1;
      %PUT YOU HAVE ENTERED AN INVALID CHOICE.;
      %PUT WOULD YOU LIKE TO TRY AGAIN? Y OR N? ;
      %INPUT AGAIN;
      %IF %UPCASE(&AGAIN)=Y %THEN %REPORT;
      %ELSE %LET BAD=1;
      %END;
   %MEND ERROR;

 /*****************************************************************/

%MACRO MENU;
   %SPACES(3)
   %PUT THE FOLLOWING REPORTS ARE AVAILABLE:;
   %SPACES(3)
   %PUT 1. A CHART OF MONTHLY SALES TO DATE BY REPRESENTATIVE.;
   %SPACES(2)
   %PUT 2. A LISTING OF MONTHLY SALES TO DATE BY REPRESENTATIVE.;
   %SPACES(3)
   %PUT PLEASE ENTER THE NUMBER OF YOUR CHOICE.;
   %INPUT CHOICE;
   %MEND MENU;

 /*****************************************************************/

%MACRO MESSAGE;
   %SPACES(2)
   %PUT THE SALES FILE HAS NOT BEEN UPDATED TODAY.;
   %PUT CALL THE INFORMATION CENTER IF YOU NEED A REPORT IMMEDIATELY.;
   %LET BAD=1;
   %MEND MESSAGE;

 /*****************************************************************/

%MACRO RC;
   %IF &SYSRC %THEN %DO;
      %PUT THE FILE YOU HAVE CHOSEN IS NOT CURRENTLY AVAILABLE.;
      %LET BAD=1;
      %END;
   %ELSE %LET BAD=0;
   %MEND RC;

 /*****************************************************************/

%MACRO SPACES(LINES);
   %DO I=1 %TO &LINES;
      %PUT %STR( );
      %END;
   %MEND SPACES;
```

Using the system of macros Begin a request for a report by calling macro REPORT:

```
%REPORT
```

You receive this menu (from macro MENU) with directions to make a choice. See **Output 19.48**.

Output 19.48 Report Selection Menu

```
THE FOLLOWING REPORTS ARE AVAILABLE:


1. A CHART OF MONTHLY SALES TO DATE BY REPRESENTATIVE.

2. A LISTING OF MONTHLY SALES TO DATE BY REPRESENTATIVE.


PLEASE ENTER THE NUMBER OF YOUR CHOICE.
```

If you enter 1, macro REPORT tests for the operating system you are using and accesses the data in the correct form for your operating system. If the macro cannot access the data set, the macro system prints this message (from macro RC):

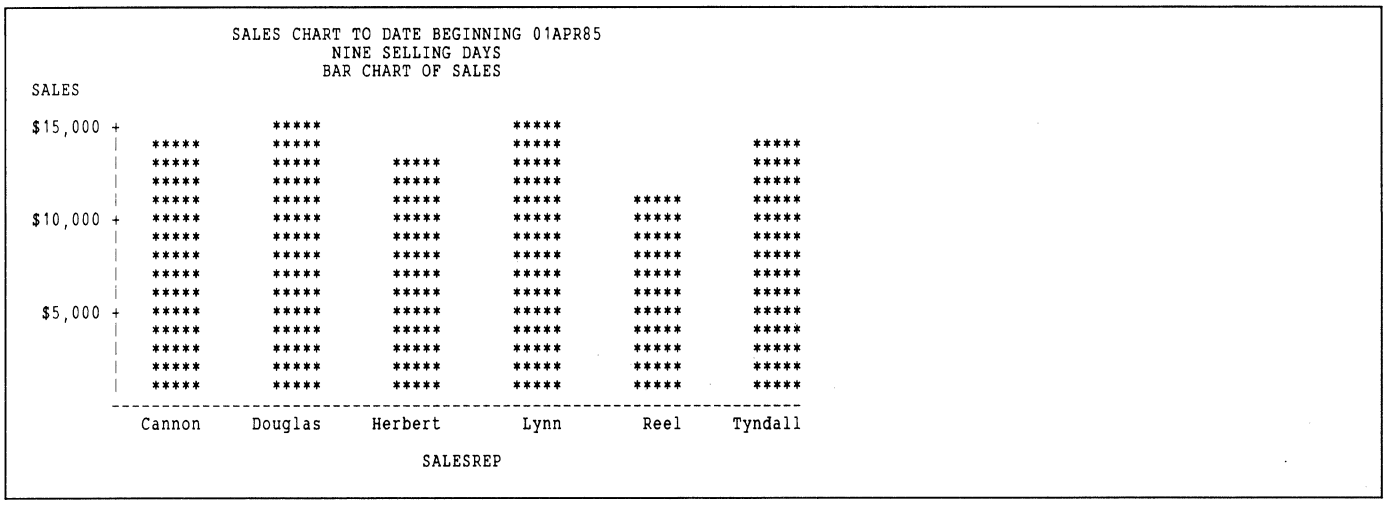
```
THE FILE YOU HAVE CHOSEN IS NOT CURRENTLY AVAILABLE.
```

If the data set is empty, macro MESSAGE prints:

```
THE SALES FILE HAS NOT BEEN UPDATED TODAY.
CALL THE INFORMATION CENTER IF YOU NEED A REPORT IMMEDIATELY.
```

If the data are available, you get a bar chart of sales. See **Output 19.49**.

Output 19.49 Bar Chart



The bar chart of sales is followed by instructions for requesting another report.

If you enter 2, the system of macros performs the same tests and provides a listing produced by the PRINT procedure followed by instructions for requesting another report. See **Output 19.50**.

Output 19.50 Listing

```
MONTHLY SALES TO DATE--REPORT BEGINNING 01APR85
             NINE SELLING DAYS
            SALESREP     SALES

            Herbert     $12,500
            Lynn        $15,000
            Reel        $11,000
            Tyndall     $13,500
            Douglas     $15,000
            Cannon      $14,000
```

If you enter 1 or 2 followed by additional characters, as in

```
2 XXXXXXX
```

the system of macros prints a warning message (from macro BUFFER):

```
EXTRA INFORMATION IGNORED:
XXXXXXX
EXECUTION WILL CONTINUE.
PLEASE BE CAREFUL TO FOLLOW THE INSTRUCTIONS.
```

and continues execution.

If you enter any other choice, the system of macros responds with this message (from macro ERROR):

```
YOU HAVE ENTERED AN INVALID CHOICE.
WOULD YOU LIKE TO TRY AGAIN? Y OR N?
```

and allows you to try again. If you enter an invalid choice the second time, the macro system prints this message (also from macro ERROR):

```
YOU HAVE EXCEEDED THE MAXIMUM NUMBER OF ATTEMPTS
   FOR THIS PROCESS.
EXECUTION WILL CEASE.
PLEASE READ YOUR INSTRUCTION SHEET OR CONSULT YOUR
   SUPERVISOR IF YOU NEED ASSISTANCE.
```

and ceases execution.

Techniques illustrated in the example This system of macros illustrates building a modular system in which many small separate pieces are combined with a driver macro. Since each small macro performs only one or two tasks, the individual pieces are easy to modify and modifying them has less chance of disturbing the whole than changing lines in a single large macro. (Of course, the macros in this example are extremely short to conserve space.) In addition, you can establish personal or group libraries of small macros and call them as needed.

Conversational macro systems like this one are often useful when combined with the SAS/FSP software product, if that product is installed at your site.

1. The OPTIONS statement at the beginning of the session causes only the text printed by the %PUT statements and the result of the PRINT and CHART procedures to appear on the screen.

2. The %GLOBAL statement in macro REPORT makes all the macro variables in the system (except macro variable AGAIN in macro ERROR) available to all the macros. Since this system of macros is designed to be used in an interactive SAS session, the global macro variables remain in existence from the first call of macro REPORT until the end of the session. To prevent the values of macro variables TIME and BAD from causing undesirable branching in subsequent calls of REPORT, macro REPORT sets their values to 0 immediately before it ceases execution (after the %STOP label).
3. The %IF &SYSENV=BACK statement tests whether you are attempting to use this system of macros as a batch job. Since this system of macros is designed for interactive use only, executing the macro as a batch job (where the automatic macro variable SYSENV has a value of BACK instead of FORE) causes a warning message to appear and execution to cease.
4. Macro SPACES prints the number of blank lines indicated by the value of macro parameter LINES. Printing blank lines is one way to control the appearance of the screen; another way is to use the %STR function with the %PUT statement to control the indentation of lines of text, as in the %PUT statements following the second %ELSE statement in macro REPORT and in macro ERROR.
5. The %IF statement illustrates conditionally calling a macro. If the value of SYSBUFFR is not null, the messages contained in macro BUFFER are printed. The semicolon after %BUFFER closes the %THEN statement; it is not part of the macro call.
6. Macro RC tests for successful allocation of the data set containing the sales information. You can use the statement

   ```
   %IF &SYSRC %THEN %DO;
   ```

   because the value of SYSRC is either 0 (false) or another number (true), never a null value. In addition, since macro variable BAD receives a value of 1 or 0 in macro RC, you can use the statement

   ```
   %IF &BAD %THEN %GO TO STOP;
   ```

   This construction has the advantage of being like the DATA step construction

   ```
   IF variable THEN statement;
   ```

   however, you must provide two non-null values (0 and non-zero) to use this form. In some cases you can use fewer macro program statements by assigning only one non-null value to the macro variable and letting a %IF statement test for a non-null value, as in

   ```
   %IF &variable¬= %THEN statement;
   ```

7. When you enter a valid value for macro variable CHOICE (1 or 2), macro CHECK tests whether the SAS data set containing the sales data is empty. The POINT= and NOBS= options in the SET statement cause SAS to retrieve the number of observations in the data set. See the SET statement in "SAS Statements Used in the DATA Step" for a description of the POINT= and NOBS= options.
8. The DATA _NULL_ step creates one of two groups of macro variables, depending on whether the SAS data set containing the sales figures has any observations.
   - If it contains no observations (NNN=0), the SYMPUT function

assigns macro variable EOF a value of E, macro MESSAGE prints a message, and the macro system ceases execution.

- If it contains observations, the INPUT function converts the value of SYSDATE to a SAS date value stored in variable NOW. The DO WHILE group calculates the number of working days (variable SELLDAYS) between the beginning of the month and the day of the most recent sales figures. Then macro variable NUM receives the value of SELLDAYS as a word; EOF receives a value of F; and macro variable DATE receives the value of the DATA step variable MONTH in the form DDMMMYY.

You can use a DATA _NULL_ step and the SYMPUT function to assign a formatted value to any macro variable. The PUT function creates a value by "writing" a variable's value in the format you specify, as though you had used a PUT statement containing a variable and a format. When you use the PUT function as the value argument of the SYMPUT function, you assign the macro variable a value "written" in the form you want. See "SAS Functions" for information on the PUT function.

9. Macro ERROR calls macro REPORT a second time if you request another try after you have entered an invalid choice. Invoking a macro from within the same macro is called *recursive execution* of a macro.

   Macro ERROR also uses the %UPCASE function to ensure that the value of macro variable AGAIN being compared to Y in the %IF statement is in uppercase, since a lowercase y does not make the comparison true. (Note: the SAS option CAPS does not affect text entered in response to a %INPUT statement.) Thus, you can enter either a lowercase or an uppercase Y as the value of AGAIN.

NOTES

1. **AOS/VS, OS, PRIMOS, VMS, and VSE**: A macro name follows the rules for a SAS data set name.

   CMS and VM/PC: A macro name follows the rules for a SAS name but can have a maximum of seven characters.

2. **AOS/VS, PRIMOS, and VMS**: The maximum length of a macro variable value is 32767 bytes.

   CMS, OS, VM/PC, and VSE: The maximum length of a macro variable value depends on the value of the MWORK= system option, which can range from 1024 to 28672 bytes. See **Usage Notes for Memory Management** for more information.

3. **VMS**: A macro name cannot contain an underscore.

   CMS and VM/PC: A macro name can contain a maximum of seven characters.

4. Unless otherwise noted, all macro-generated statements are displayed with the MPRINT option.

5. The RUN statement in "SAS Statements Used Anywhere" includes an example in which a TITLE statement is affected by a step boundary.

6. **OS**: In interactive (not display manager) sessions under TSO, a subset of log information is routed to the terminal by default. Use the LOG(*) operand in the SAS command to route the complete log to the terminal, as in

```
SAS LOG(*)
```

where SAS is the command you use to begin an interactive session. See the *SAS Companion for OS Operating Systems and TSO* for more information.

7. **AOS/VS, PRIMOS, and VMS**: Memory is handled differently, and these options are ignored.

8. **CMS, OS, VM/PC, and VSE**: You can also recode variables in this way with a SELECT group. Macro CODESEL performs the same function as macro CODEIT but uses a SELECT group:

```
%MACRO CODESEL(START=,STOP=,COUNT=,OLD=,NEW=);
   %*OPERATING SYSTEMS: CMS, OS, VM/PC, VSE;
   SELECT;
   %DO I=&START+&COUNT %TO &STOP %BY &COUNT;
      WHEN(&START<=&OLD<&I) &NEW=&START;
      %LET START=&I;
      %END;
   WHEN(&START<=&OLD) &NEW=&START;
   END;
%MEND CODESEL;
%CODESEL(START=30,STOP=59,COUNT=3,OLD=AGEYR,NEW=AGE3)
```

SAS PROCEDURES

SAS® Descriptive Procedures
SAS® Reporting Procedures
SAS® Utility Procedures
The APPEND Procedure
The BMDP Procedure
The BROWSE Procedure
The CALENDAR Procedure
The CHART Procedure
The COMPARE Procedure
The CONTENTS Procedure
The CONVERT Procedure
The COPY Procedure
The CORR Procedure
The DATASETS Procedure
The EDITOR Procedure
The FORMAT Procedure
The FORMS Procedure
The FREQ Procedure
The MEANS Procedure
The OPTIONS Procedure
The PDS Procedure
The PDSCOPY Procedure
The PLOT Procedure
The PRINT Procedure
The PRINTTO Procedure
The RELEASE Procedure
The SORT Procedure
The SOURCE Procedure
The SUMMARY Procedure
The TABULATE Procedure
The TAPECOPY Procedure
The TAPELABEL Procedure
The TIMEPLOT Procedure
The TRANSPOSE Procedure
The UNIVARIATE Procedure
The XCOPY Procedure

Chapter 20

SAS® Descriptive Procedures

Introduction

The following SAS descriptive procedures compute univariate or bivariate descriptive statistics such as means, sums, standard deviations, and correlations:

MEANS	computes means and other univariate descriptive statistics
UNIVARIATE	computes univariate statistics including quantiles and draws distributional plots.

SUMMARY computes univariate descriptive statistics within groups of observations. The groups are defined by variables in a CLASS statement. The statistics are written to a SAS data set but not printed.

TABULATE prints complex tables of descriptive statistics

CORR computes bivariate correlations and other measures of association for quantitative variables.

Other procedures that compute decriptive statistics include:

CHART draws bar, block, pie, and star charts of counts, means, or sums. CHART is described under **Reporting**.

FREQ computes frequency distributions for categorical variables and does multi-way crosstabulations. FREQ computes a wide variety of descriptive and inferential statistics and is documented in the *SAS User's Guide: Statistics*.

Comparison of Procedures

Table 20.1 indicates the statistics available from each procedure and summarizes some other important features.

Table 20.1 SAS Descriptive Procedures and Their Features

Statistic	MEANS	UNIVARIATE	SUMMARY	TABULATE	CORR
number of missing values	x	x	x	x	
number of nonmissing values	x	x	x	x	x
sum of weights	x	x	x	x	
mean	x	x	x	x	x
sum	x	x	x	x	x
minimum	x	x	x	x	x
maximum	x	x	x	x	x
range	x	x	x	x	
uncorrected sum of squares	x	x	x	x	

continued on next page

Table 20.1 *Continued*

Statistic	MEANS	UNIVARIATE	SUMMARY	TABULATE	CORR
corrected sum of squares	x	x	x	x	
variance	x	x	x	x	
standard deviation	x	x	x	x	x
standard error	x	x	x	x	
coefficient of variation	x	x	x	x	
skewness	x	x			
kurtosis	x	x			
Student's *t*	x	x	x	x	
prob>*t*	x	x	x	x	
median		x			x
quartiles		x			
mode		x			
Pearson correlation					x
Spearman correlation					x
Kendall correlation					x
Hoeffding's D					x
Other Features					
printed output	yes	yes	no	yes	yes
output to a SAS data set	yes	yes	yes	no	yes
CLASS statement	no	no	yes	yes	no
BY statement	yes	yes	yes	yes	yes

Efficiency

Most descriptive statistics are computed with one pass over the data set, but PROC MEANS uses two passes if skewness and kurtosis are requested.

Quantiles, including the median, require time proportional to $n\log(n)$ for large sample sizes, so PROC UNIVARIATE may require more time than other descriptive procedures. UNIVARIATE also requires more storage because the data are held in core.

If you need to compute statistics for each of several groups of observations, you can use any of the above procedures with a BY statement to specify the groups. However, BY-group processing requires sorting the data set, which can be expensive for large data sets. SUMMARY and TABULATE can produce statistics across a classification without sorting. PROC SUMMARY creates an output data set but prints nothing, while PROC TABULATE prints tables of descriptive statistics in hierarchical crosstabulations but produces no output data set.

Keywords and Formulas

A standardized set of keywords is used to refer to the univariate descriptive statistics in SAS procedures. These keywords are used in SAS statements to request statistics for printing or storing in an output data set.

The following notations are used where summation is over all nonmissing values:

x_i	the ith nonmissing observation on the variable
w_i	the weight associated with x_i if a WEIGHT statement is specified, otherwise 1.
n	the number of nonmissing observations
$\bar{x}$	$= \Sigma w_i x_i / \Sigma w_i$
d	$= n$ if the option VARDEF=N is specified, $= n - 1$ if VARDEF=DF, $= \Sigma w_i$ if VARDEF=WEIGHT or WGT, or $= \Sigma w_i - 1$ if VARDEF=WDF
s^2	$= \Sigma w_i (x_i - \bar{x})^2 / d$
z_i	$= (x_i - \bar{x})/s$ standardized variables.

The formulas and standard keywords for each statistic are given below. In some formulas a keyword is used to designate the corresponding statistic.

N	number of nonmissing observations
NMISS	number of missing observations
MIN	minimum value
MAX	maximum value
RANGE	MAX-MIN, the range
SUM	$\Sigma w_i x_i$, the weighted sum
SUMWEIGHT	Σw_i, the sum of weights
MEAN	$\bar{x}$, the arithmetic mean
USS	$\Sigma w_i x_i^2$, the uncorrected sum of squares
CSS	$\Sigma w_i (x_i - \bar{x})^2$, the sum of squares corrected for the mean
VAR	s^2, the variance
STD	s, the standard deviation

STDERR	$s/\sqrt{(n)}$, the standard error of the mean
CV	$100s/\bar{x}$, the percent coefficient of variation
SKEWNESS	$\Sigma z_i^3 \times n/(n-1)(n-2)$, measures sidedness
KURTOSIS	$\Sigma z_i^4 \times n(n+1)/(n-1)(n-2)(n-3) - 3(n-1)^2/(n-2)(n-3)$, measures heaviness of tails
T	$t=\bar{x}\sqrt{(n)}/s$, Student's t for H_0: population mean=0
PRT	the two-tailed p-value for Student's t with $n-1$ degrees of freedom, the probability under the null hypothesis of obtaining an absolute value of t greater than the t value observed in this sample
MEDIAN	the middle value when the x_i are arranged in order of value and n is odd; the mean of the two middle values when n is even.
QUARTILE	the values such that one quarter of the x_i are larger and one quarter of the x_i are smaller
MODE	the most frequent value of x_i

Data Requirements for Univariate Statistics

Statistics are reported as missing if they cannot be computed. N and NMISS do not require any nonmissing observations. SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require only one nonmissing observation. The requirements for other statistics are as follows:

- VAR, STD, STDERR, CV, T, and PRT require at least 2 observations.
- SKEWNESS requires at least 3 observations.
- KURTOSIS requires at least 4 observations.
- SKEWNESS, KURTOSIS, T, and PRT require STD>0.
- SKEWNESS and KURTOSIS are not computed if a WEIGHT statment is used.
- CV requires MEAN$\neq$0.

Background

This section provides a brief introduction to some of the statistical concepts necessary for interpreting the output of SAS procedures for descriptive statistics. For a more thorough discussion, consult an introductory statistics textbook such as Moore (1979), Ott (1977), or Snedecor and Cochran (1967).

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, we may have as the universe all of the students at a particular school and two populations of interest: one of height values and one of weight values. Or the universe could be 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 failing.

A population of values can be described in terms of its *cumulative distribution function*, which gives the proportion of the population less than each possible value. A discrete population can also be described by a *probability function* giving the proportion of the population equal to each possible value. A continuous pop-

ulation can often be described by a *density function*, which is the derivative (see a calculus book) of the cumulative distribution function. A density function can be approximated by a histogram giving the proportion of the population lying within each of a series of intervals of values, such as produced by the CHART procedure. A probability density function is like a histogram with an infinite number of infinitely small intervals.

In technical literature, the term *distribution* used without qualification generally refers to the cumulative distribution function. In informal writing, *distribution* sometimes means the density function instead. Often the word *distribution* is used simply to refer to an abstract population of values rather than some concrete population. Thus the statistical literature refers to many types of abstract distributions such as normal distributions, exponential distributions, Cauchy distributions, and so on. When a phrase such as *normal distribution* is used, it frequently does not matter whether the cumulative distribution function or the density function is intended.

It may be expedient to describe a population in terms of a few measures that summarize interesting features of the distribution. Such a 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, 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.

We write $E(x)$ to denote the mean of a population of values symbolized by x, such as height, where E stands for *expected value*. We can also consider expected values of derived functions of the original values. For example, if x represents height, then $E(x^2)$ is the expected value of height squared, that is, the mean value of the population obtained by squaring every value in the population of heights.

Samples and Statistics

It is often impossible to measure all of the values in a population. A collection of measured values is called a *sample*. A mathematical function of a sample of values is called a *statistic*. A statistic is to a sample as a parameter is to a population. It is customary to denote statistics by Roman letters and parameters by Greek letters. For example, the population mean is often written as μ, whereas the sample mean is written as $\bar{x}$. The branch of mathematics called *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 thirty students from the high school, the mean height for those thirty 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, there are many statistics that can be used to 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 statistic are the same. In other cases, the formula for a parameter may be different from that of the most commonly used estimator. It should not be assumed that the most commonly used estimator is the best estimator in all applications.

Measures of Location

Measures of location include the mean, the mode, and the median. These measures can generally be thought of as describing the center of a distribution. In the definitions below, notice that if the entire sample is changed by adding a fixed amount to each observation, then these measures of location are shifted by the same fixed amount.

Mean As discussed above, the population mean $\mu = E(x)$ is usually estimated by the sample mean:

$$\bar{x} = \Sigma x_i/n \quad .$$

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.

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. If there is a tie for most-often-occuring sample value, UNIVARIATE reports the lowest such value. If the population is continuous, then all sample values occur once and the sample mode has little use. In such cases, population modes can be estimated via nonparametric density estimation; see the METHOD=DENSITY option in the CLUSTER procedure in the *SAS User's Guide: Statistics*.

Quantiles

Quantiles, including percentiles, 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% of the measurements below it and $(100-p)$% above it. The median is the 50th percentile. Since it may not be possible to divide your data so that you get exactly the desired percentile, a more precise definition is used (see the UNIVARIATE procedure).

The upper quartile of a distribution is the value below which 75% of the measurements fall (the 75th percentile). Twenty-five percent of the measurements fall below the lower quartile value. Selected percentiles and quartiles are calculated by the UNIVARIATE procedure. The RANK procedure can be used to calculate any desired quantiles.

Example: Quantiles and Measures of Location

The data in the following example are artificially generated by a call to a pseudo-random-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 variables are used in the macro %FORMGEN to produce value labels for PROC FORMAT. The resulting format is used with PROC CHART to display the values of the statistics on a histogram.

```
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;

PROC UNIVARIATE;
   OUTPUT OUT=LOCATION MEAN=MEAN MODE=MODE MEDIAN=MEDIAN
   Q1=Q1 Q3=Q3 P5=P5 P10=P10 P90=P90 P95=P95 MAX=MAX;
PROC PRINT;

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;

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; FOOTNOTE;
```

Output 20.1 Example of Quantiles and Measures of Location

EXAMPLE OF QUANTILES AND MEASURES OF LOCATION 1

UNIVARIATE

VARIABLE=X

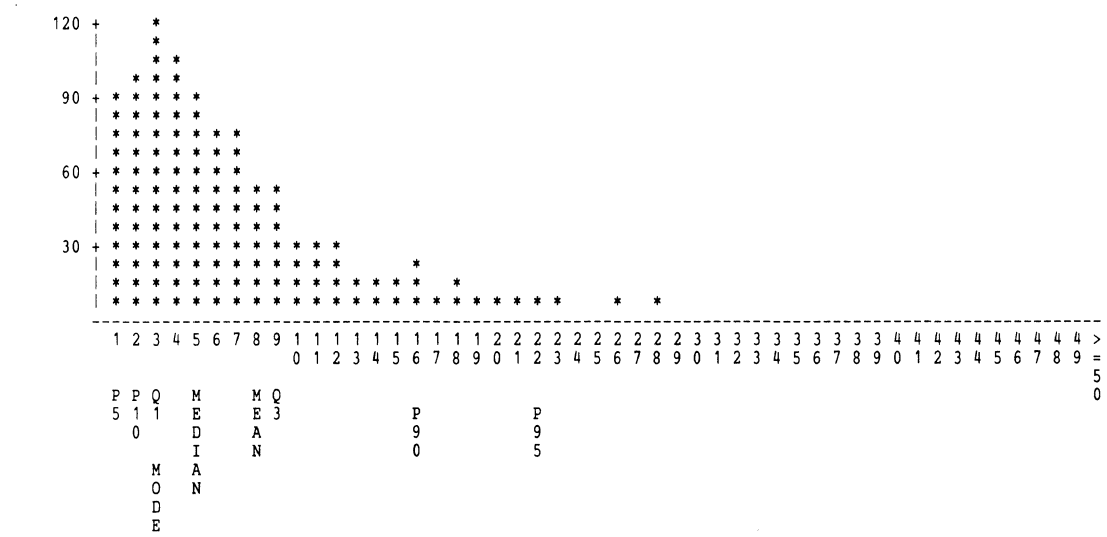
MOMENTS				QUANTILES(DEF=4)			EXTREMES	
N	1000	SUM WGTS	1000	100% MAX	62	99%	37.9897	LOWEST HIGHEST
MEAN	7.605	SUM	7605	75% Q3	9	95%	21.95	0 44
STD DEV	7.3817	VARIANCE	54.4895	50% MED	5	90%	16	0 44
SKEWNESS	2.73039	KURTOSIS	11.1871	25% Q1	3	10%	2	0 57
USS	112271	CSS	54435	0% MIN	0	5%	1	0 61
CV	97.0637	STD MEAN	0.23343			1%	0	0 62
T:MEAN=0	32.5794	PROB>|T|	0.0001	RANGE	62			
SGN RANK	244778	PROB>|S|	0.0001	Q3-Q1	6			
NUM ^= 0	989			MODE	3			

EXAMPLE OF QUANTILES AND MEASURES OF LOCATION 2

OBS	MEAN	MAX	P5	P10	Q1	MEDIAN	Q3	P90	P95	MODE
1	7.605	62	1	2	3	5	9	16	21.95	3

EXAMPLE OF QUANTILES AND MEASURES OF LOCATION 3

FREQUENCY BAR CHART



P5 = 5TH PERCENTILE
P10 = 10TH PERCENTILE
P90 = 90TH PERCENTILE
P95 = 95TH PERCENTILE
Q1 = 1ST QUARTILE
Q3 = 3RD QUARTILE

Measures of Variability

Another group of statistics important for studying the distribution of a population measures the *variability*, or spread, of values. In the definitions given below, 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. The values of these statistics are changed, however, if each observation in the sample is multiplied by a constant.

Range The sample range is the difference between the largest and smallest values in the sample. For many populations, at least in statistical theory, the range is infinite, so the sample range may not tell you much about the population. The sample range tends to increase as the sample size increases. If all sample values are multiplied by a constant, the sample range is multiplied by the same constant.

Interquartile range The interquartile range is the difference between the upper and lower quartiles. If all sample values are multiplied by a constant, the sample interquartile range is multiplied by the same constant.

Variance The population variance, usually denoted by σ^2 when it is clear what population is being considered, is the expected value of the squared difference of the values from the population mean:

$$\sigma^2 = E(x-\mu)^2 \quad .$$

The sample variance, denoted s^2, is usually computed as

$$s^2 = \Sigma\,(x_i-\bar{x})^2 \,/\, (n-1) \quad .$$

The difference between a value and the mean is called a *deviation from the mean*. Thus the variance is the mean squared deviation from the mean. 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, 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 is used.

Standard deviation The standard deviation is the square root of the variance, or root-mean-square deviation from the mean, in either population or 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, 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, 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 dis-

tributions, positive deviations may tend to be larger in magnitude than negative deviations, or vice versa. *Skewness* is a measure of the tendency of the deviations to be larger in one direction than in the other. For example, the data in the last example are skewed to the right.

Population skewness is defined as

$$E(x-\mu)^3/\sigma^3 \quad .$$

Since 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.

SAS calculates the sample skewness as

$$n/(n-1)(n-2)\Sigma_{i=1}^{n}(x_i-\bar{x})^3/s^3 \quad .$$

Skewness can be positive or negative and is unbounded.

Kurtosis The heaviness of the tails of a population 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

$$E(x-\mu)^4/\sigma^4 - 3 \quad ,$$

although some statisticians omit the subtraction of 3. Since 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 positive infinity, inclusive. If m_3 represents population skewness and m_4 represents population kurtosis, then $m_4 \geq m_3^2-2$.

SAS uses the formula below to calculate sample kurtosis:

$$\text{Kurtosis}=n(n+1)/(n-1)(n-2)(n-3)\ \Sigma_{i=1}^{n}(x_i-\bar{x})^4/s^4 - 3(n-1)(n-1)/(n-2)(n-3).$$

There is a myth in the statistical literature that kurtosis measures the *peakedness* of a density. This myth stubbornly persists despite repeated debunking (Kaplansky 1945; Ali 1974; Johnson, Tietjen, and Beckman 1980).

Sample skewness and kurtosis are rather unreliable estimators of the corresponding parameters in small samples. Trust them only if you have a very large sample. However, large values of skewness or kurtosis may merit attention even in small samples because such values indicate that statistical methods based on normality assumptions (see below) may be inappropriate.

The Normal Distribution

One especially important family of theoretical distributions is the *normal* or *Gaussian* distribution. A normal density 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 standard deviation. Approximately 68% of the values in a normal population are within one standard deviation of the population mean; approximately 95% of the values are within two standard deviations of the mean; and about 99.7% are

within three standard deviations. Use of the term normal to describe this particular 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 non-normality and, if necessary, check your sample for evidence of non-normality.

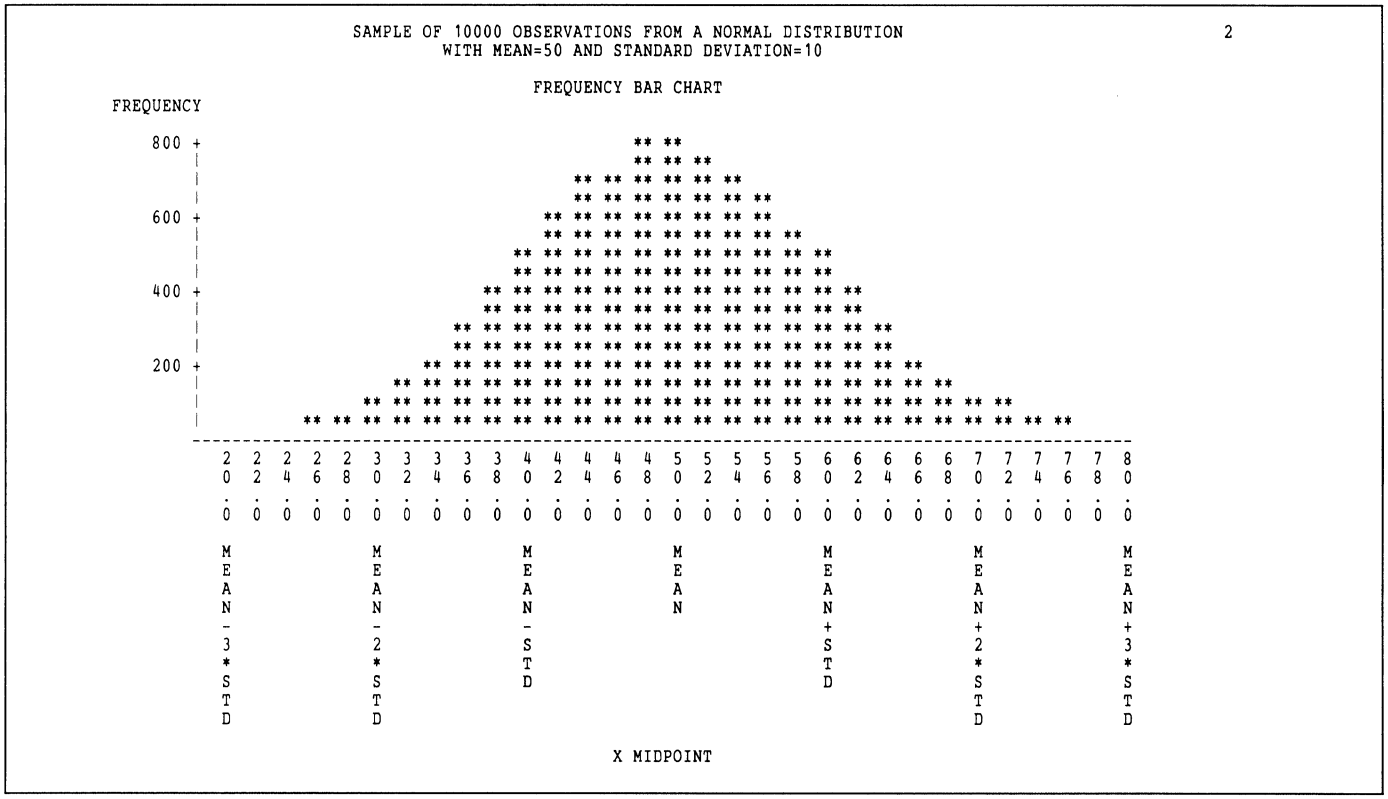
Example: Normal Distribution

The following data are sampled from a normal distribution with mean 50 and standard deviation 10.

```
TITLE 'SAMPLE OF 10000 OBSERVATIONS FROM A NORMAL DISTRIBUTION';
TITLE2 'WITH MEAN=50 AND STANDARD DEVIATION=10';
DATA NORMAL;
   DROP N;
   DO N=1 TO 10000;
      X=10*RANNOR(53429)+50;
      OUTPUT;
      END;

PROC UNIVARIATE;
   VAR X;

PROC FORMAT PRINT;
   PICTURE MSD
      20='20.0 MEAN-3*STD' (NOEDIT)
      30='30.0 MEAN-2*STD' (NOEDIT)
      40='40.0 MEAN-STD  ' (NOEDIT)
      50='50.0 MEAN      ' (NOEDIT)
      60='60.0 MEAN+STD  ' (NOEDIT)
      70='70.0 MEAN+2*STD' (NOEDIT)
      80='80.0 MEAN+3*STD' (NOEDIT)
   OTHER='00.0           '
   ;

PROC CHART;
   VBAR X/MIDPOINTS=20 TO 80 BY 2;
   FORMAT X MSD.;
```

Output 20.2 Sample from a Normal Distribution

```
                    SAMPLE OF 10000 OBSERVATIONS FROM A NORMAL DISTRIBUTION                    1
                          WITH MEAN=50 AND STANDARD DEVIATION=10

                                    UNIVARIATE

VARIABLE=X

                MOMENTS                                  QUANTILES(DEF=4)                EXTREMES

N             10000  SUM WGTS       10000      100% MAX    88.7939     99%     73.2131     LOWEST    HIGHEST
MEAN         49.901  SUM            499010      75% Q3       56.73     95%     66.4972    8.94372    82.1407
STD DEV      10.0244 VARIANCE      100.488      50% MED     49.8617     90%     62.8185    11.6688    83.3165
SKEWNESS  0.0128066  KURTOSIS  -0.0164395      25% Q1      43.0938     10%     37.1346     14.221    83.4689
USS       25905927  CSS            1004782       0% MIN    8.94372      5%     33.5476    15.1369    86.1095
CV           20.0885 STD MEAN      0.100244                              1%     26.5005    15.5051    88.7939
T:MEAN=0     497.797 PROB>|T|       0.0001      RANGE       79.8502
SGN RANK  25002500  PROB>|S|       0.0001      Q3-Q1       13.6361
NUM ^= 0     10000                            MODE        35.8226
```



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.

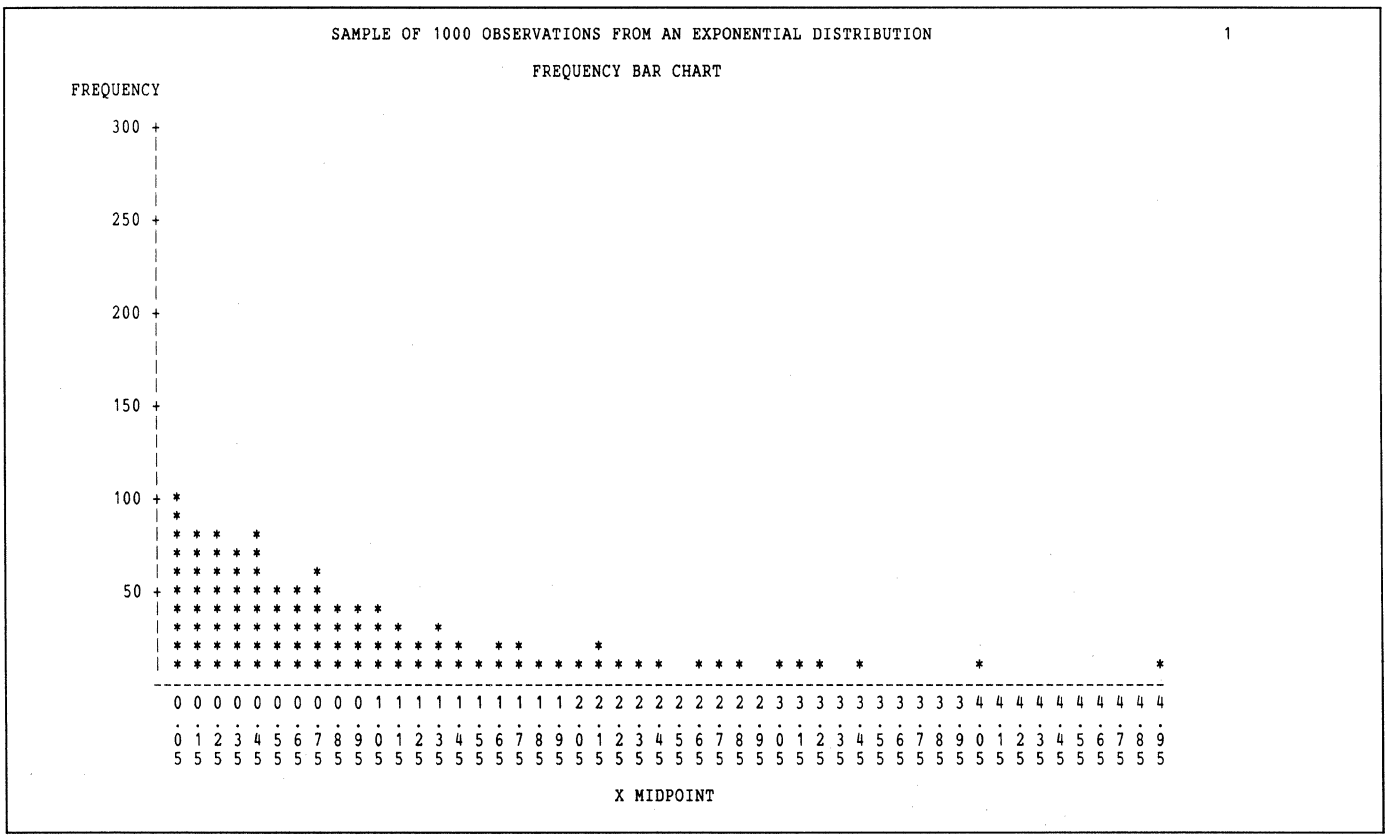
Example: Sampling Distributions

The following DATA step creates a sample of 1000 observations from an exponential distribution generated by the RANEXP function. 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 1.04.

This is an example of a non-normal 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.

```
TITLE 'SAMPLE OF 1000 OBSERVATIONS FROM AN EXPONENTIAL DISTRIBUTION';
DATA EXPO;
   DROP N;
   DO N=1 TO 1000;
      X=RANEXP(18746363);
      OUTPUT;
      END;

PROC CHART;
   VBAR X/AXIS=0 TO 300 BY 50 MIDPOINTS=0.05 TO 4.95 BY .1;

PROC UNIVARIATE;
```

Output 20.3 Sample from an Exponential Distribution



```
                    SAMPLE OF 1000 OBSERVATIONS FROM AN EXPONENTIAL DISTRIBUTION                2

                                          UNIVARIATE

VARIABLE=X

                MOMENTS                                  QUANTILES(DEF=4)                EXTREMES

N              1000  SUM WGTS        1000      100% MAX    6.63907     99%      5.0584      LOWEST    HIGHEST
MEAN        1.01176  SUM           1011.76       75% Q3     1.35737     95%      3.14373  0.00055441     5.7822
STD DEV     1.04371  VARIANCE      1.08933       50% MED   0.689502     90%      2.37899  0.00124782    6.00679
SKEWNESS    1.96963  KURTOSIS       4.8015       25% Q1    0.294032     10%     0.102007  0.00222294     6.1354
USS         2111.91  CSS           1088.25        0% MIN 0.00055441      5%    0.0519024  0.00253325    6.60897
CV          103.158  STD MEAN   0.0330051                               1%   0.00956337  0.00396732    6.63907
T:MEAN=0    30.6548  PROB>|T|       0.0001       RANGE      6.63851
SGN RANK     250250  PROB>|S|       0.0001       Q3-Q1      1.06334
NUM ^= 0       1000                            MODE    0.00055441
```

The next DATA step draws 1000 different samples from the same population as the previous DATA step. Each sample contains 10 observations. PROC MEANS computes the mean of each sample. In the data set created by PROC MEANS, each observation represents the mean of a sample of 10 observations from an exponential distribution. Thus, the data set is a sample from the sampling distribution of the mean for an exponential population.

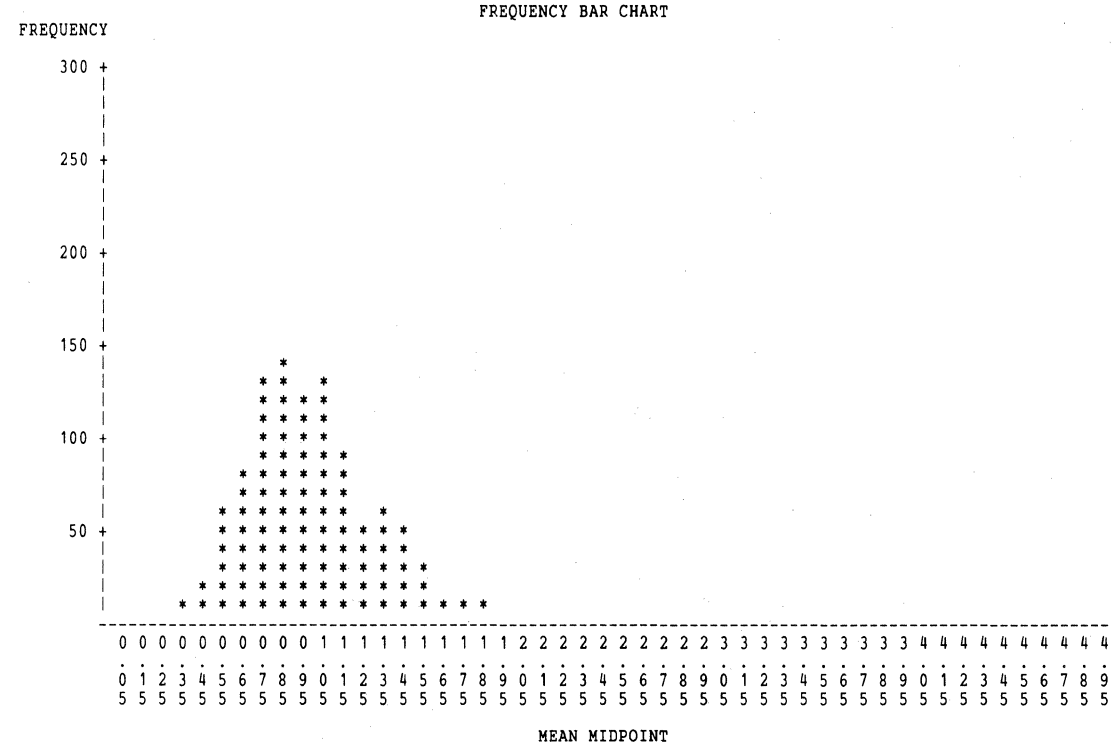
PROC UNIVARIATE displays statistics for this sample of means. Notice that the mean of the sample of means is .99, almost the same as the mean of the original population. Theoretically, the standard deviation of the sampling distribution is $\sigma/\sqrt{(n)}=1.00/\sqrt{(10)}=.32$, whereas the standard deviation of this sample from the sampling distribution is .31. The skewness (.55) and kurtosis ($-.01$) are closer to zero in the sample from the sampling distribution than in the original sample from the exponential distribution, since the sampling distribution is closer to a normal distribution than is the original exponential distribution. A histogram of the 1000 sample means is shown by PROC CHART. The shape of the histogram is much closer to a bell-like normal density, but it is still distinctly lopsided.

```
TITLE '1000 SAMPLE MEANS WITH 10 OBSERVATIONS 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 NOPRINT;
   OUTPUT OUT=MEAN10 MEAN=MEAN;
   VAR X;
   BY SAMPLE;

PROC CHART;
   VBAR MEAN/AXIS=0 TO 300 BY 50 MIDPOINTS=0.05 TO 4.95 BY .1;

PROC UNIVARIATE;
   VAR MEAN;
```

Output 20.4 Sample Drawn from an Exponential Distribution

1000 SAMPLE MEANS WITH 10 OBSERVATIONS PER SAMPLE 1
DRAWN FROM AN EXPONENTIAL DISTRIBUTION

FREQUENCY BAR CHART



1000 SAMPLE MEANS WITH 10 OBSERVATIONS PER SAMPLE 2
DRAWN FROM AN EXPONENTIAL DISTRIBUTION

UNIVARIATE

VARIABLE=MEAN

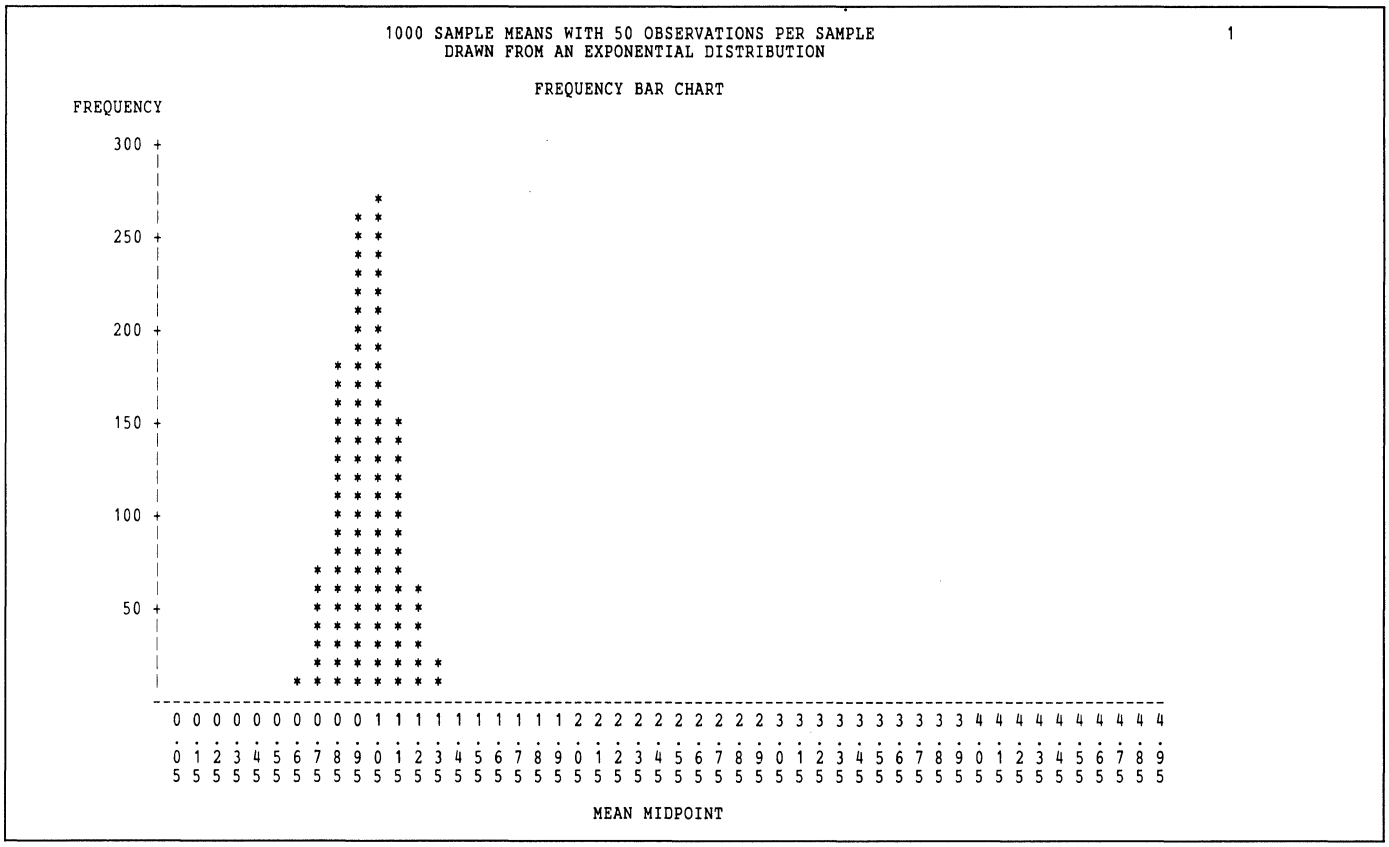
MOMENTS				QUANTILES(DEF=4)				EXTREMES	
N	1000	SUM WGTS	1000	100% MAX	2.0539	99%	1.82841	LOWEST	HIGHEST
MEAN	0.990686	SUM	990.686	75% Q3	1.18169	95%	1.55725	0.256069	1.90337
STD DEV	0.307326	VARIANCE	0.0944496	50% MED	0.956152	90%	1.41709	0.300903	1.91309
SKEWNESS	0.545753	KURTOSIS	-.00609601	25% Q1	0.763899	10%	0.621714	0.343799	1.92148
USS	1075.81	CSS	94.3551	0% MIN	0.256069	5%	0.552574	0.364235	1.93335
CV	31.0216	STD MEAN	0.00971852			1%	0.431807	0.380808	2.0539
T:MEAN=0	101.938	PROB>|T|	0.0001	RANGE	1.79783				
SGN RANK	250250	PROB>|S|	0.0001	Q3-Q1	0.417787				
NUM ^= 0	1000			MODE	1.02664				

In the following DATA step, the size of each sample 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 from the skewness.

```
TITLE '1000 SAMPLE MEANS WITH 50 OBSERVATIONS 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 NOPRINT;
   OUTPUT OUT=MEAN50 MEAN=MEAN;
   VAR X;
   BY SAMPLE;

PROC CHART;
   VBAR MEAN/AXIS=0 TO 300 BY 50 MIDPOINTS=0.05 TO 4.95 BY .1;

PROC UNIVARIATE;
   VAR MEAN;
```

Output 20.5 An Exponential Distribution with Increased Sample Size



```
                    1000 SAMPLE MEANS WITH 50 OBSERVATIONS PER SAMPLE                    2
                         DRAWN FROM AN EXPONENTIAL DISTRIBUTION

                                      UNIVARIATE

VARIABLE=MEAN

                MOMENTS                                QUANTILES(DEF=4)                 EXTREMES

N              1000  SUM WGTS        1000      100% MAX   1.45496     99%      1.34677      LOWEST    HIGHEST
MEAN       0.996797  SUM           996.797      75% Q3     1.08669     95%      1.23167    0.584558    1.37536
STD DEV    0.138154  VARIANCE   0.0190865      50% MED    0.996023    90%      1.17981    0.670252    1.37801
SKEWNESS   0.190621  KURTOSIS   -0.143872      25% Q1     0.896917    10%     0.814876     0.67143    1.39192
USS         1012.67  CSS            19.0675     0% MIN    0.584558     5%     0.780326     0.67221      1.403
CV          13.8598  STD MEAN  0.00436881                               1%     0.701112    0.680194    1.45496
T:MEAN=0    228.162  PROB>|T|       0.0001      RANGE      0.870399
SGN RANK    250250  PROB>|S|       0.0001      Q3-Q1       0.18977
NUM ^= 0      1000                           MODE       0.584558
```

Testing Hypotheses

The purpose of the statistical methods 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 aforementioned universe of students. 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:

```
DATA X;
   TITLE 'DEVIATIONS FROM NORMAL WEIGHT';
   INPUT X @@;
   CARDS;
-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, you can write this hypothesis, called the *null* hypothesis, as H_0: $\mu=0$. The other two hypotheses, called *alternative* hypotheses, are that the students are underweight on the average, H_1: $\mu<0$, and that the students are overweight on the average, H_1: $\mu>0$.

The null hypothesis is so called not because the hypothesized parameter value is zero, but because the hypothesis specifies a specific value for the parameter. The null hypotheses is not credible because the mean of a continuous variable is almost never exactly equal to any known value. The null hypothesis can be considered a straw man to be knocked down by statistical evidence, and you can decide between the alternative hypotheses according to which way it 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 $\bar{x}<0$, decide on H_1: $\mu<0$
- if $\bar{x}=0$, decide on H_0: $\mu=0$
- if $\bar{x}>0$, decide on H_2: $\mu>0$.

The trouble with this approach is that there may be a high probability of making an incorrect decision. If H_0 is true, 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, there may be nearly a 50% chance that $\bar{x}$ will be greater

than zero in repeated sampling, so the chances of incorrectly choosing H_2would also be nearly 50%. Thus, you have a high probability of making an error if $\bar{x}$ is near zero. In such cases, there is not enough evidence to make a confident decision, so the best response may be to reserve judgment until you can obtain more evidence.

The question is, how far from zero must $\bar{x}$ be for you to be able to make a confident decision? The answer can be obtained by considering the sampling distribution of $\bar{x}$. If x has a roughly normal distribution, then $\bar{x}$ has an approximately normal sampling distribution. The mean of the sampling distribution of $\bar{x}$ is μ. Assume temporarily that σ, the standard deviation of x, is known to be 12. Then the standard error of $\bar{x}$ for samples of nine observations is $\sigma/\sqrt{(n)}=12/\sqrt{(9)}=4$. You know that about 95% of the values from a normal distribution are within two standard deviations of the mean, so about 95% of the possible samples of nine x values have a sample mean $\bar{x}$ between $0-2 \times 4$ and $0+2 \times 4$, or between -8 and 8. Consider the chances of making an error with the following decision rule:

- if $\bar{x}<-8$, decide on H_1: $\mu<0$
- if $-8<=\bar{x}<=8$, reserve judgment
- if $\bar{x}>8$, decide on H_2: $\mu>0$

If H_0 is true, then in about 95% of the possible samples $\bar{x}$ will be between the *critical values* -8 and 8, so you will reserve judgment. In these cases the statistical evidence is not strong enough to fell the straw man. In the other 5% of the samples you will make an error; in 2.5% you will incorrectly choose H_1, and in 2.5% you will incorrectly choose H_2.

If H_1 is true, you will make an error only if $\bar{x}$ is greater than 8. To calculate the chances of this occurrence you need to know the actual value of μ, but the chance of an error cannot exceed 2.5%. Similarly, the chance of an incorrect decision if $H_2$is true cannot exceed 2.5%.

Thus, using this decision rule, you will make an incorrect decision in at most about 5% of the possible samples. 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 *significance* level of the statistical test. In this example, an $\bar{x}$ value less than -8 or greater than 8 is said to be *statistically significant* at the 5% level. You can adjust the significance level according to your needs by choosing different critical values. For example, critical values of -4 and 4 would produce a significance level of about 32%, while -12 and 12 would give a significance level of about 0.3%.

The decision rule stated above 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, you could use a *one-tailed* test:

- if $\bar{x}<=8$, reserve judgment
- if $\bar{x}>8$, decide on H_2: $\mu>0$.

For this one-tailed test, the significance level is 2.5%, half that of the two-tailed test.

The probability of rejecting the null hypothesis if it is false is called the *power* of the statistical test. The power depends on the true value of the parameter. In the example, assume the population mean is 4. The power for detecting H_2 is the probability of getting a sample mean greater than 8. The critical value 8 is one standard error higher than the population mean 4. The chance of getting

a value at least standard deviation greater than the mean from a normal distribution is about 16%, so the power for detecting the alternative hypothesis H_2 is about 16%. If the population mean were 8, the power for H_2 would be 50%, whereas a population mean of 12 would yield a power of about 84%.

The smaller the significance level, the less the chance of making an incorrect decision, but the higher the chance of having to reserve judgment. In choosing a significance level, you should consider the resulting power for various alternatives of interest.

Student's t distribution Unfortunately, you could not really use the decision rule described above because you do not know the value of σ. You could, however, use s as an estimate of σ. Consider the following statistic:

$$t = (\bar{x}-\mu_0)/(s/\sqrt{(n)}) ,$$

which is the difference between the sample mean and the hypothesized mean μ_0 divided by the estimated standard error of the mean. A t statistic for the null hypothesis that $\mu=0$ can be obtained along with related statistics from PROC MEANS:

```
PROC MEANS DATA=X N MEAN STD STDERR T PRT;
```

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 $t<-2.3$, decide on H_1: $\mu<0$
- if $-2.3<=t<=2.3$, reserve judgment
- if $t>2.3$, decide on H_2: $\mu>0$.

The value 2.3 was obtained from a table of Student's t distribution in a statistics text to give a significance level of 5% for 8 (that is, 9−1) degrees of freedom. If you do not have a statistics text handy, you can print a table of the t distribution using a DATA step:

```
DATA _NULL_;
   TITLE;
   FILE PRINT;
   ARRAY P P1-P7;
   RETAIN P1 .10 P2 .05 P3 .025 P4 .01 P5 .005 P6 .0025 P7 .001;
   PUT '              TABLE OF STUDENT''S T DISTRIBUTION'
     / '              --------------------------------'
    // '  DF               TWO-TAILED SIGNIFICANCE LEVEL'
     / '  --   ----------------------------------------------------'
     / '     ' @;
   DO OVER P;
      PC=100*P;
      PUT PC BEST7. '%' @;
      END;
   PUT /;
   DO DF=1 TO 30, 35 TO 50 BY 5, 60 TO 100 BY 10, 200 TO 1000 BY 100;
      PUT DF 4. @;
      DO OVER P;
```

```
      B=BETAINV(P,DF/2,.5);
      F=DF*(1-B)/B;
      T=SQRT(F);
      PUT T 8.3 @;
      END;
   PUT;
   END;
```

Output 20.6 Output of a Table Showing the T Distribution

DEVIATIONS FROM NORMAL WEIGHT 1

VARIABLE	N	MEAN	STANDARD DEVIATION	STD ERROR OF MEAN	T	PR>|T|
X	9	8.55555556	11.75915719	3.91971906	2.18	0.0606

2

TABLE OF STUDENT'S T DISTRIBUTION

DF	TWO-TAILED SIGNIFICANCE LEVEL						
	10%	5%	2.5%	1%	0.5%	0.25%	0.1%
1	6.314	12.706	25.452	63.657	127.321	254.647	636.619
2	2.920	4.303	6.205	9.925	14.089	19.962	31.599
3	2.353	3.182	4.177	5.841	7.453	9.465	12.924
4	2.132	2.776	3.495	4.604	5.598	6.758	8.610
5	2.015	2.571	3.163	4.032	4.773	5.604	6.869
6	1.943	2.447	2.969	3.707	4.317	4.981	5.959
7	1.895	2.365	2.841	3.499	4.029	4.595	5.408
8	1.860	2.306	2.752	3.355	3.833	4.334	5.041
9	1.833	2.262	2.685	3.250	3.690	4.146	4.781
10	1.812	2.228	2.634	3.169	3.581	4.005	4.587
11	1.796	2.201	2.593	3.106	3.497	3.895	4.437
12	1.782	2.179	2.560	3.055	3.428	3.807	4.318
13	1.771	2.160	2.533	3.012	3.372	3.735	4.221
14	1.761	2.145	2.510	2.977	3.326	3.675	4.140
15	1.753	2.131	2.490	2.947	3.286	3.624	4.073
16	1.746	2.120	2.473	2.921	3.252	3.581	4.015
17	1.740	2.110	2.458	2.898	3.222	3.543	3.965
18	1.734	2.101	2.445	2.878	3.197	3.510	3.922
19	1.729	2.093	2.433	2.861	3.174	3.481	3.883
20	1.725	2.086	2.423	2.845	3.153	3.455	3.850
21	1.721	2.080	2.414	2.831	3.135	3.432	3.819
22	1.717	2.074	2.405	2.819	3.119	3.412	3.792
23	1.714	2.069	2.398	2.807	3.104	3.393	3.768
24	1.711	2.064	2.391	2.797	3.091	3.376	3.745
25	1.708	2.060	2.385	2.787	3.078	3.361	3.725
26	1.706	2.056	2.379	2.779	3.067	3.346	3.707
27	1.703	2.052	2.373	2.771	3.057	3.333	3.690
28	1.701	2.048	2.368	2.763	3.047	3.321	3.674
29	1.699	2.045	2.364	2.756	3.038	3.310	3.659
30	1.697	2.042	2.360	2.750	3.030	3.300	3.646
35	1.690	2.030	2.342	2.724	2.996	3.258	3.591
40	1.684	2.021	2.329	2.704	2.971	3.227	3.551
45	1.679	2.014	2.319	2.690	2.952	3.203	3.520
50	1.676	2.009	2.311	2.678	2.937	3.184	3.496
60	1.671	2.000	2.299	2.660	2.915	3.156	3.460
70	1.667	1.994	2.291	2.648	2.899	3.137	3.435
80	1.664	1.990	2.284	2.639	2.887	3.122	3.416
90	1.662	1.987	2.280	2.632	2.878	3.111	3.402
100	1.660	1.984	2.276	2.626	2.871	3.102	3.390
200	1.653	1.972	2.258	2.601	2.839	3.062	3.340
300	1.650	1.968	2.253	2.592	2.828	3.049	3.323
400	1.649	1.966	2.250	2.588	2.823	3.043	3.315
500	1.648	1.965	2.248	2.586	2.820	3.039	3.310
600	1.647	1.964	2.247	2.584	2.817	3.036	3.307
700	1.647	1.963	2.246	2.583	2.816	3.034	3.304
800	1.647	1.963	2.246	2.582	2.815	3.033	3.303
900	1.647	1.963	2.245	2.581	2.814	3.032	3.301
1000	1.646	1.962	2.245	2.581	2.813	3.031	3.300

In the current example, the value of the t statistic is 2.18, so using a 5% significance level you must reserve judgment. If you had elected to use a 10% significance level, 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 from the value specified in the null 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 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 the significance level of the test, the null hypothesis can be rejected. In the example, the two-tailed p-value, labeled PR> |T| on the PROC MEANS printout, is .0606, so the null hypothesis could be rejected at the 10% significance level but not at the 5% level.

A p-value is a measure of the strength of the evidence against the null hypothesis. The smaller the p-value, the stronger the evidence for rejecting the null hypothesis.

Bivariate Measures

Bivariate statistics measure the degree of dependence between two variables. Bivariate measures for continuous variables are available in PROC CORR. These include correlations, rank correlations, and measures of concordance. These measures are discussed in detail in the CORR procedure description. Bivariate measures for categorical data include chi-square tests of independence and various measures of association. These are discussed in the chapter "SAS Categorical Procedures" in the *SAS User's Guide: Statistics*.

REFERENCES

Ali, M.M. (1974), "Stochastic Ordering and Kurtosis Measure," JASA, 69, 543-545.

Fisher, R.A. (1973), *Statistical Methods for Research Workers*, 14th Edition, New York: Hafner Publishing Company.

Johnson, M.E., Tietjen, G.L., and Beckman, R.J. (1980), "A New Family of Probability Distributions With Applications to Monte Carlo Studies," JASA, 75, 276-279.

Kaplansky, I. (1945), "A Common Error Concerning Kurtosis," JASA, 40, 259-263.

Moore, D.S. (1979), *Statistics: Concepts and Controversies*, San Francisco: W.H. Freeman and Company.

Ott, L. (1977), *An Introduction to Statistical Methods and Data Analysis*, North Scituate, Mass.: Duxbury Press.

Snedecor, G.W. and Cochran, W.C. (1967), *Statistical Methods*, 6th Edition, Ames, Iowa: Iowa State University Press.

Chapter 21

SAS® Reporting Procedures

Introduction

Reporting procedures produce a display of information. The display can be a listing of data, an organized display of data, or a graphical display such as a bar chart or a scatter plot. Most descriptive and statistical procedures also produce reports to display results, but the procedures in this section are specialized for reporting.

PRINT	prints values from a SAS data set.
FORMS	prints mailing list labels or other data laid out in repetitive forms.
CHART	charts frequencies and other statistics with bar charts and other pictorial representations.
PLOT	plots variables in a scatter diagram.
CALENDAR	prints data in the form of a summary or schedule calendar.
TIMEPLOT	plots one or more variables over time intervals.

In addition to these procedures, several reporting methods are covered outside this chapter.

PUT statements
: are used to program custom reports. (See "Data Step Applications.")

COMPUTAB
: is a row-by-column financial reporting procedure available with SAS/ETS software and documented in the *SAS/ETS User's Guide*.

GCHART and GPLOT
: produce graphical reports like CHART and PLOT on graphics devices. These are available with SAS/GRAPH software and are documented in the *SAS/GRAPH User's Guide*.

The PRINT Procedure

The PRINT procedure is an easy procedure for listing the data in a SAS data set. The simplest invocation

```
PROC PRINT;
```

prints the most recently created SAS data set. Each variable in the data set forms a column of the report; each observation of the data set forms a row of the report. If you want variables to form rows and observations to form columns, use PROC TRANSPOSE to change variables into observations and vice versa before you print the data set.

PROC PRINT is able to

- use FORMAT statements to associate variables with formats
- use TITLE and FOOTNOTE statements to define up to ten title and ten footnote lines
- use LINESIZE= and PAGESIZE= system options to control size
- print any subset of the variables
- use an ID variable to identify observations
- divide the report into sections when there are too many variables to fit across the page
- separate groups of observations according to BY groups
- print totals and other statistics for all observations and BY groups
- print variable labels as column headings.

You tell PROC PRINT what to do, not how to do it. While other SAS reporting methods can require a lot of programming, no programming is required with PROC PRINT.

But the PRINT procedure may be too limited for some applications. It cannot select subsets of the observations to print, and almost all computations must be done ahead of time in DATA steps.

The following example creates a SAS data set and then prints it twice using PROC PRINT, once in a very simple manner and again using formats, totals, variable labels, and an ID variable.

```
DATA A;
   INPUT YEAR SALES COST;
   PROFIT=SALES-COST;
   CARDS;
1981 12132 11021
1982 19823 12928
1983 16982 14002
1984 18432 14590
;

PROC PRINT DATA=A;
   TITLE 'SIMPLE PROC PRINT REPORT';
PROC PRINT DATA=A SPLIT='*';
   LABEL SALES='SALES FOR *  YEAR'
      COST='TOTAL COST'
      PROFIT='PROFIT BEFORE *    TAXES';
   FORMAT SALES COST PROFIT DOLLAR10.2;
   ID YEAR;
   SUM SALES COST PROFIT;
   TITLE 'PROC PRINT WITH TOTALS, FORMATS, LABELS, AND ID VARIABLE';
```

Output 21.1 Two PROC PRINT Reports

```
                          SIMPLE PROC PRINT REPORT                          1

                OBS    YEAR    SALES    COST     PROFIT

                 1     1981    12132    11021     1111
                 2     1982    19823    12928     6895
                 3     1983    16982    14002     2980
                 4     1984    18432    14590     3842
```

```
                PROC PRINT WITH TOTALS, FORMATS, LABELS, AND ID VARIABLE    2

                   YEAR     SALES FOR        TOTAL        PROFIT BEFORE
                              YEAR           COST            TAXES

                   1981    $12,132.00    $11,021.00       $1,111.00
                   1982    $19,823.00    $12,928.00       $6,895.00
                   1983    $16,982.00    $14,002.00       $2,980.00
                   1984    $18,432.00    $14,590.00       $3,842.00
                           ==========    ==========       ==========
                           $67,369.00    $52,541.00       $14,828.00
```

The FORMS Procedure

The FORMS procedure is designed to print mailing labels and other types of repetitive forms. This example adds labeling variables to the data in order to identify the fields on the form units. The forms are laid out three units across the page.

```
DATA B;
   SET A;
   TYEAR ='YEAR  = ';
   TSALES='SALES = ';
   TCOST ='COST  = ';
PROC FORMS W=20 D=1 NA=3 BETWEEN=5 DATA=B;
   FORMAT SALES COST COMMA10.2 YEAR 10.;
   LINE 1 TYEAR YEAR;
   LINE 2 TSALES SALES;
   LINE 3 TCOST COST;
```

Output 21.2 PROC FORMS Report

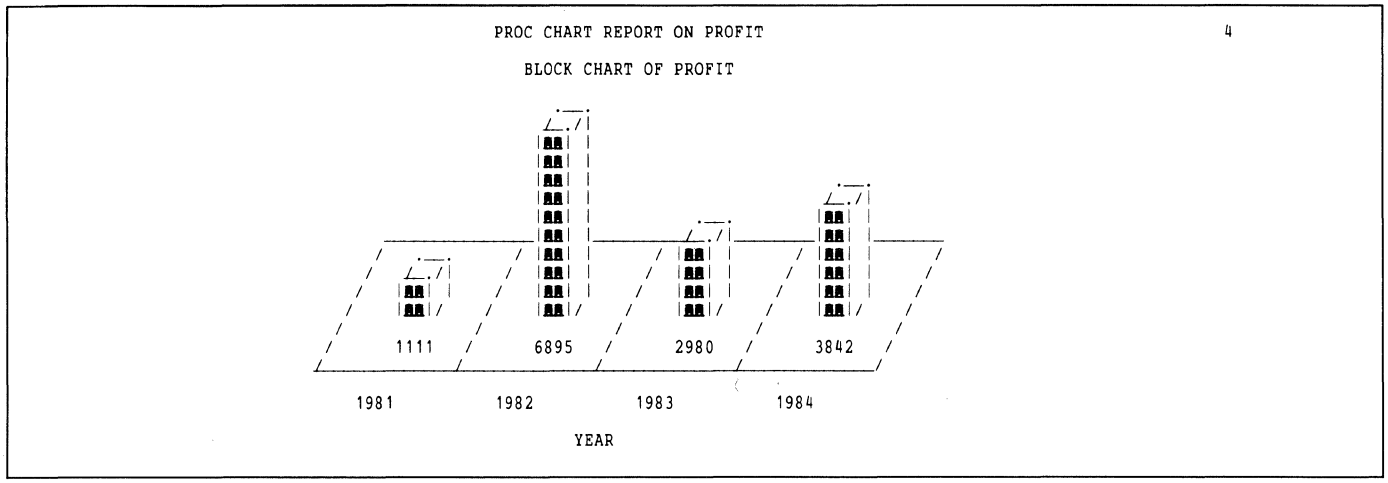
```
                                                                            3

YEAR  =        1981      YEAR  =        1982      YEAR  =        1983
SALES =   12,132.00      SALES =   19,823.00      SALES =   16,982.00
COST  =   11,021.00      COST  =   12,928.00      COST  =   14,002.00

YEAR  =        1984
SALES =   18,432.00
COST  =   14,590.00
```

The CHART Procedure

The CHART procedure draws a variety of charts: horizontal and vertical bar charts, block charts, pie charts, and star charts. The procedure can chart counts, percentages, means, and sums. **Output 21.3** is a block chart of the PROFIT variable from the PRINT example.

```
PROC CHART DATA=A;
   TITLE 'PROC CHART REPORT ON PROFIT';
   BLOCK YEAR / SUMVAR=PROFIT DISCRETE SYMBOL='XOA';
```

Output 21.3 PROC CHART Report



The PLOT Procedure

The PLOT procedure draws scatter plots. The following example plots three variables against YEAR. The three variables appear on the same plot, identified by different plotting characters.

```
PROC PLOT DATA=A;
   TITLE 'PROC PLOT REPORT: P=PROFIT S=SALES C=COST';
   PLOT PROFIT*YEAR='P'
        SALES*YEAR='S'
        COST*YEAR='C' / OVERLAY VPOS=20 HPOS=32;
```

Output 21.4 PROC PLOT Report

```
PROC PLOT REPORT: P=PROFIT S=SALES C=COST                    5

 PLOT OF PROFIT*YEAR     SYMBOL USED IS P
 PLOT OF SALES*YEAR      SYMBOL USED IS S
 PLOT OF COST*YEAR       SYMBOL USED IS C

 PROFIT |
        |
        |
 20000  +          S
        |                              S
        |                    S
        |
 15000  +                              C
        |                    C
        |S          C
        |C
 10000  +
        |
        |          P
        |
  5000  +                              P
        |                    P
        |P
     0  +
        -+---------+---------+---------+-
        1981      1982      1983      1984

                     YEAR
```

The CALENDAR Procedure

Use the CALENDAR procedure if you have either daily data on some activity or data representing events, each of which is identified by a date. In both cases, CALENDAR displays the data in the form of a monthly calendar. You can specify formats for variables, missing value options, and customized borders for the calendar. You can also specify the calendar to cover all the days in a month or weekdays only (Monday-Friday). This procedure is useful for two main applications:

- daily reporting, such as daily tallies of sales or usage. You can also collect sums and means.
- scheduling, making calendars of events.

Output 21.5 is a summary calendar that displays telephone calls received by a library during daytime and evening hours. Variables are printed using picture formats created by PROC FORMAT, and sums and means of the variables appear in the legend.

```
DATA TEL;
   INPUT DATE : DATE. DAY NIGHT @@;
   CARDS;
 1OCT85 33 48  2OCT85 29 10  3OCT85 18 35  4OCT85 18 23
 5OCT85 36 45  6OCT85 25 44  7OCT85 28 40  8OCT85 29 49
 9OCT85 31 17 10OCT85 20 .  11OCT85 19 29 12OCT85 30 11
13OCT85 28 49 14OCT85 27 46 15OCT85 20 36 16OCT85 27 48
17OCT85 20 39 18OCT85 27 46 19OCT85 20 36 20OCT85 27 48
21OCT85 33 48 22OCT85 29 10 23OCT85 18 35 24OCT85 18 23
25OCT85 36 45 26OCT85 25 44 27OCT85 28 40 28OCT85 29 49
29OCT85 31 17 30OCT85 20 .  31OCT85 19 29
;
PROC FORMAT;
   PICTURE DDD OTHER='000 DAY';
   PICTURE NNN OTHER='000 NIGHT';
PROC CALENDAR HEADER=SMALL LEGEND;
   ID DATE;
   VAR DAY NIGHT;
   SUM DAY NIGHT;
   MEAN DAY NIGHT;
   LABEL DAY='DAYTIME CALLS' NIGHT='NIGHT CALLS';
   FORMAT DAY DDD. NIGHT NNN. ;
   TITLE 'TELEPHONE CALLS RECEIVED AT CITY LIBRARY';
```

Output 21.5 PROC CALENDAR Report

TELEPHONE CALLS RECEIVED AT CITY LIBRARY 6

OCTOBER 1985

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1 33 DAY 48 NIGHT	2 29 DAY 10 NIGHT	3 18 DAY 35 NIGHT	4 18 DAY 23 NIGHT	5 36 DAY 45 NIGHT
6 25 DAY 44 NIGHT	7 28 DAY 40 NIGHT	8 29 DAY 49 NIGHT	9 31 DAY 17 NIGHT	10 20 DAY .	11 19 DAY 29 NIGHT	12 30 DAY 11 NIGHT
13 28 DAY 49 NIGHT	14 27 DAY 46 NIGHT	15 20 DAY 36 NIGHT	16 27 DAY 48 NIGHT	17 20 DAY 39 NIGHT	18 27 DAY 46 NIGHT	19 20 DAY 36 NIGHT
20 27 DAY 48 NIGHT	21 33 DAY 48 NIGHT	22 29 DAY 10 NIGHT	23 18 DAY 35 NIGHT	24 18 DAY 23 NIGHT	25 36 DAY 45 NIGHT	26 25 DAY 44 NIGHT
27 28 DAY 40 NIGHT	28 29 DAY 49 NIGHT	29 31 DAY 17 NIGHT	30 20 DAY .	31 19 DAY 29 NIGHT		

LEGEND	SUM	MEAN
DAYTIME CALLS	798	25.7419
NIGHT CALLS	1039	35.8276

The TIMEPLOT Procedure

The TIMEPLOT procedure produces a plot and a listing of observations in the data set similar to that produced by the PLOT and PRINT procedures. However, a plot produced by TIMEPLOT has these features:

- The vertical axis always represents the sequence of observations in the data set; thus, if the observations are arranged chronologically, the vertical axis represents time.
- The horizontal axis contains values of the variables you are examining.
- A plot can occupy more than one page.
- Each observation appears on a separate line; no observations are hidden as can occur with PROC PLOT.
- Each observation in the plot is accompanied by a listing of the values plotted.

The following example overlays plots of U.S. and world cotton production on the same axes. A vertical reference line appears at the mean value of each variable, and a horizontal line connects the values within each observation.

```
DATA COTTON;
   INPUT YEAR US WORLD @@;
   LABEL US='US COTTON PRODUCTION'
      WORLD='WORLD COTTON PRODUCTION';
   CARDS;
1940 12.6 31.2 1975  8.3 54.0
1950 10.0 30.6 1976 10.6 57.4
1960 14.2 46.2 1977 14.4 63.5
1965 15.0 55.0 1978 10.9 60.2
```

```
1970 10.2 53.6 1979 14.6 65.7
1972 13.7 62.9 1980 11.1 65.5
1973 13.0 63.3 1981 15.6 70.8
1974 11.5 64.3 1982 12.0 67.4
;
PROC SORT;
   BY YEAR;
PROC TIMEPLOT;
   PLOT US WORLD / OVERLAY REF=MEAN(US WORLD) HILOC POS=64;
   ID YEAR;
   TITLE 'TRENDS IN COTTON PRODUCTION';
   TITLE2 'SELECTED YEARS 1940-1982';
   TITLE4 '(MILLIONS OF BALES)';
RUN;
```

Output 21.6 PROC TIMEPLOT Report

```
                                TRENDS IN COTTON PRODUCTION                                    7
                                  SELECTED YEARS 1940-1982

                                     (MILLIONS OF BALES)

YEAR      US COTTON      WORLD COTTON       MIN                                              MAX
        PRODUCTION        PRODUCTION       8.3                                             70.8
                                           *-------------------------------------------------------*
1940         12.60             31.20       |    U-----------------W                       |       |
1950         10.00             30.60       |  U-|-----------------W                       |       |
1960         14.20             46.20       |    | U-------------------------------W       |       |
1965         15.00             55.00       |    |  U--------------------------------------W |       |
1970         10.20             53.60       |  U-|-----------------------------------------W |       |
1972         13.70             62.90       |    | U-----------------------------------------|-----W |
1973         13.00             63.30       |    |U------------------------------------------|-----W |
1974         11.50             64.30       |   U|------------------------------------------|------W|
1975          8.30             54.00       |U---|-------------------------------------------W |       |
1976         10.60             57.40       |  U-|--------------------------------------------W       |
1977         14.40             63.50       |    | U-----------------------------------------|------W|
1978         10.90             60.20       |   U|-------------------------------------------|--W   |
1979         14.60             65.70       |    | U-----------------------------------------|--------W|
1980         11.10             65.50       |   U|-------------------------------------------|--------W|
1981         15.60             70.80       |    |  U----------------------------------------|------------W|
1982         12.00             67.40       |    U-----------------------------------------|----------W  |
                                           *-------------------------------------------------------*
```

Reports Written with PUT Statements

The PUT statement is a program statement used in a DATA step. Reports written with PUT statements use program statements to create the form of the report. You specify the column positions and program every calculation.

Using PUT statements for report writing is documented in detail in "DATA Step Applications." (See also the **PUT Statement** in "SAS Statements Used in the DATA Step.") **Output 21.7** is an example of a report produced with PUT statements.

You want to print the observations as columns and the variables as rows in the report. This is most conveniently done with the N=PS feature (see the **FILE Statement** in "SAS Statements Used in the DATA Step") and by computing column positions with program statements. The variable C indexes column positions corresponding to observations from the data and a column for the total.

```
DATA _NULL_;
   FILE PRINT N=PS;
   TITLE 'PUT STATEMENT REPORT';

   *---BRING IN THREE OBSERVATIONS PER PAGE---;
   DO C=25 TO 61 BY 12;     *---C WILL BE THE PRINTING COLUMN---;
      SET A END=EOF;        *---BRING IN AN OBSERVATION HERE----;
```

```
*---COMPUTE GROSS PROFIT AND ACCUMULATE TOTALS---;
TOTSALES+SALES;
TOTCGS+COST;
TOTPROF+PROFIT;

*---PRINT OUT A COLUMN FOR THIS MONTH---;
PUT #5
   'YEAR'                @C YEAR    10.
// 'GROSS REVENUE'       @C SALES   10.2
 / 'COST OF GOODS SOLD'  @C COST     10.2
 /                       @C '=========='
 / 'GROSS PROFIT'        @C PROFIT 10.2;

*---AT END OF DATA---;
IF EOF THEN DO;
   C=C+12;
   *---COMPUTE MARGIN AND PRINT TOTALS---;
   PCTMARG=100*TOTPROF/TOTCGS;
   PUT #5 @C '   TOTAL'
      // @C TOTSALES  10.2
       / @C TOTCGS    10.2
       / @C '=========='
       / @C TOTPROF   10.2
      // 'PROFIT MARGIN PERCENT OF COST' @C PCTMARG 10.2;
   END;
END;
```

Output 21.7 PUT Statement Report

```
                                    PUT STATEMENT REPORT                                    8


YEAR                          1981        1982        1983        1984       TOTAL

GROSS REVENUE              12132.00    19823.00    16982.00    18432.00    67369.00
COST OF GOODS SOLD         11021.00    12928.00    14002.00    14590.00    52541.00
                          ==========  ==========  ==========  ==========  ==========
GROSS PROFIT                1111.00     6895.00     2980.00     3842.00    14828.00

PROFIT MARGIN PERCENT OF COST                                                  28.22
```

Chapter 22

SAS® Utility Procedures

A utility procedure performs a specific type of intermediate processing or data manipulation. Utility applications include interfacing, sorting, editing, reformatting, and copying data sets, rerouting output from other procedures, and much more. Utility procedures usually produce little or no output and require few control language statements. Some utility procedures are designed to handle applications on specific operating systems. Each procedure description includes a list of operating systems under which the procedure can be used. Following is a list and brief description of the utility procedures fully described in this chapter. If you are unable to find a utility procedure to do exactly what you want, you can probably program a DATA step to handle the task.

APPEND	appends data from one SAS data set to the end of another SAS data set.
BMDP	interfaces with a BMDP program.
BROWSE	reads a SAS data set when read-only access is permitted.
CONTENTS	describes the contents of a SAS data library or specified members of the library.
CONVERT	converts files from other systems or earlier releases of the SAS System to SAS data sets.
COPY	copies a SAS data library or selected members of the library.
DATASETS	manages a SAS data library.
EDITOR	edits directly in disk-format SAS data sets.
FORMAT	defines output formats for value labeling.
OPTIONS	lists the current values of all SAS system options.
PDS	lists, deletes, and renames members of OS partitioned data sets.
PDSCOPY	copies OS partitioned data sets containing load modules.
PRINTTO	specifies a destination for printed output produced by SAS procedures.
RELEASE	releases unused space at the end of an OS disk data set.
SORT	sorts a SAS data set according to one or more variables.
SOURCE	prints or unloads the contents of an OS partitioned data set containing card-image records or a VSE source statement sublibrary.

TAPECOPY	copies files from one or more tape volumes to one output tape under OS batch, TSO, and CMS.
TAPELABEL	lists the label information on a standard-labeled tape under OS batch, TSO, and CMS.
TRANSPOSE	turns a SAS data set on its side, changing variables into observations and observations into variables.
XCOPY	changes a SAS data library to transport format or changes a transport format library to a SAS data library under OS, CMS, and VSE.

Chapter 23

The APPEND Procedure

Operating systems: All

ABSTRACT

The APPEND procedure adds the observations from one SAS data set to the end of another SAS data set. PROC APPEND can be used only with disk format data sets.

INTRODUCTION

Refer to the chapters "Introduction to the DATA Step" and "SAS Files" to familiarize yourself with SAS data sets and related terminology.

Often you need to add new observations to a SAS data set. If you use a DATA step to concatenate two data sets, the SAS System must process all the observations in both data sets to create a new one. The APPEND procedure bypasses the processing of data in the original data set and adds new observations directly to the end of the original data set.

SPECIFICATIONS

PROC APPEND BASE=*SASdataset options***;**

The PROC statement is the only statement associated with the APPEND procedure.

PROC APPEND Statement

PROC APPEND BASE=*SASdataset options;*

BASE=*SASdataset*
OUT=*SASdataset*
: names the data set to which you want to add observations. If APPEND cannot find an existing data set with this name, it creates a new data set. In other words, you can use PROC APPEND to create a data set by specifying a new data set name in BASE=. OUT= is equivalent to BASE=. Either BASE= or OUT= must be specified.

The APPEND procedure can be used with these options:

DATA=*SASdataset*
NEW=*SASdataset*
: names the SAS data set containing observations you want to add to the end of the SAS data set specified by BASE=. If DATA= is omitted, the most recently created SAS data set is used. NEW= is equivalent to DATA=.

FORCE
: causes truncation of values if variables in the DATA= data set have longer lengths than the corresponding variables in the BASE= data set. FORCE also causes variables in the DATA= data set to be dropped if there are not corresponding variables in the BASE= data set. DATA= variables that do not match BASE= variables are assigned missing values in the output (BASE=) data set when FORCE is in effect. A mismatch will occur if a DATA= variable name or variable type (character or numeric) does not match a corresponding variable name or variable type in the BASE= data set. If DATA= variables are too long or do not match BASE= variables, the SAS System prints a message on the log and stops processing unless FORCE is specified.

DETAILS

Usage Notes

When the APPEND procedure has finished processing, the BASE= data set in its modified form becomes the current SAS data set.

Using PROC APPEND can be more efficient than using a SET statement in the DATA step to concatenate two data sets, especially if the BASE= data set is large. Consider the following example:

```
DATA LARGE;
   SET LARGE SMALL;
```

To perform this concatenation the SAS System must read all observations in both LARGE and SMALL. With the APPEND procedure you can add the observations in the SMALL data set to the end of the LARGE data set without reading the observations in LARGE. For example, the statement

```
PROC APPEND BASE=LARGE DATA=SMALL;
```

offers a much more efficient way to concatenate the two data sets. The procedure is very useful if observations are frequently added to a SAS data set (for example, in production programs that are constantly appending data to a journal-type data set).

If all variables in the BASE= data set have the same length and relative position as the variables in the DATA= data set and if all variables are in both data sets, the procedure executes faster. You can use PROC CONTENTS with the

POSITION option for each data set to see if the variable lengths and positions are identical.

Note: since APPEND operates on the BASE= data set in update mode, the BASE= data set cannot be a tape-format data set. Also, if there is a system failure or if some other type of interruption occurs while the procedure is executing, the BASE= data set may not be properly updated; it is possible that not all, perhaps none, of the observations will be added to the BASE= data set.

Output Data Set

All observations from the DATA= data set are added to the BASE= data set. If the BASE= data set does not exist, APPEND creates a new SAS data set consisting of the observations in the DATA= data set. The output data set (BASE=) becomes the current SAS data set.

If there are variables in the BASE= data set that are not in the DATA= data set, those variables have missing values in the new observations.

Messages pertaining to procedure processing appear in the SAS log, but no printed output is produced.

EXAMPLES

Appending an Update File: Example 1

An update file called ADD contains information on average temperatures in cities around the country. It is appended to a master file, EXP.MASTER. PROC PRINT lists the master file before and after the execution of PROC APPEND.

```
DATA ADD;
   INPUT CITY $15. MONTH $15. TEMP;
   CARDS;
RALEIGH         JULY            77.5
RALEIGH         JANUARY         40.5
MIAMI           AUGUST          82.9
MIAMI           JANUARY         67.2
LOS_ANGELES     AUGUST          69.5
LOS_ANGELES     JANUARY         54.5
PROC PRINT DATA=EXP.MASTER;
   TITLE 'MASTER BEFORE ADD DATA APPENDED';
PROC APPEND BASE=EXP.MASTER DATA=ADD;
PROC PRINT DATA=EXP.MASTER;
   TITLE 'MASTER AFTER ADD DATA APPENDED';
RUN;
```

Output 23.1 The BASE= Data Set before and after the PROC APPEND Step

```
       S A S   L O G    OS SAS 5.03        XXX/XXX JOB AP1      STEP S AS      PROC

NOTE: THE JOB AP1 HAS BEEN RUN UNDER RELEASE 5.03 OF SAS AT SAS INSTITUTE DATA CENTER (02718001).

NOTE: CPUID   VERSION = 11  SERIAL = 221547  MODEL = 3083 .

NOTE: SAS OPTIONS SPECIFIED ARE:
      SORT=4

1          DATA ADD;
2             INPUT CITY $15. MONTH $15. TEMP;
3          CARDS;

NOTE: DATA SET WORK.ADD HAS 6 OBSERVATIONS AND 3 VARIABLES. 453 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.07 SECONDS AND 92K.

11         PROC PRINT DATA=EXP.MASTER;
12            TITLE 'MASTER BEFORE ADD DATA APPENDED';

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 144K AND PRINTED PAGE 1.

13         PROC APPEND BASE=EXP.MASTER DATA=ADD;

NOTE: EXP.MASTER HAS 6 OBSERVATIONS BEFORE APPENDING.
NOTE: 6 OBSERVATION(S) ADDED.
NOTE: THE PROCEDURE APPEND USED 0.10 SECONDS AND 512K.

14         PROC PRINT DATA=EXP.MASTER;
15            TITLE 'MASTER AFTER ADD DATA APPENDED';

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 144K AND PRINTED PAGE 2.

NOTE: THE DATA STATEMENT USED 0.05 SECONDS AND 84K.

20         RUN;

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C.  27511-8000
```

MASTER BEFORE ADD DATA APPENDED 1

OBS	CITY	MONTH	TEMP
1	HONOLULU	AUGUST	80.7
2	HONOLULU	JANUARY	72.3
3	BOSTON	JULY	73.3
4	BOSTON	JANUARY	29.2
5	DULUTH	JULY	65.6
6	DULUTH	JANUARY	8.5

MASTER AFTER ADD DATA APPENDED 2

OBS	CITY	MONTH	TEMP
1	HONOLULU	AUGUST	80.7
2	HONOLULU	JANUARY	72.3
3	BOSTON	JULY	73.3
4	BOSTON	JANUARY	29.2
5	DULUTH	JULY	65.6
6	DULUTH	JANUARY	8.5
7	RALEIGH	JULY	77.5
8	RALEIGH	JANUARY	40.5
9	MIAMI	AUGUST	82.9
10	MIAMI	JANUARY	67.2
11	LOS_ANGELES	AUGUST	69.5
12	LOS_ANGELES	JANUARY	54.5

Using PROC APPEND with FORCE: Example 2

This example illustrates one use of the FORCE option. Suppose you are gathering weather data each day and storing it in a SAS data set named ACCUM.WEATHER.

By running PROC PRINT on ACCUM.WEATHER, you see that it already contains this information:

```
OBS      DATE      TEMP      SUNHRS

 1      01JAN84     35        9.30
 2      02JAN84     34        9.33
 3      03JAN84     37        9.35
 4      04JAN84     38        9.39
```

The following program appends the weather information accumulated the next day to ACCUM.WEATHER and prints a copy of the results:

```
DATA DAILY;
   INPUT DATE MMDDYY8. TEMP SUNHRS PRECIP;
   CARDS;
01-05-84 35 9.40 0.00
;
PROC APPEND BASE=ACCUM.WEATHER FORCE;
PROC PRINT;
   FORMAT DATE DATE7.;
   TITLE 'UP-TO-DATE WEATHER INFORMATION';
RUN;
```

Notice that the data set DAILY contains the additional variable PRECIP for which there is no corresponding variable in the BASE= data set. With FORCE in effect, however, the observation is appended to ACCUM.WEATHER, but the value for PRECIP is dropped. Also notice that the DATA= option is not necessary on the PROC statement because DAILY is the most recently created SAS data set before the PROC step executes.

Output 23.2 FORCE Option Causes PRECIP Value To Be Dropped

```
       S A S   L O G    OS SAS 5.03          XXX/XXX JOB AP2      STEP  SAS  PROC

NOTE: THE JOB AP2 HAS BEEN RUN UNDER RELEASE 5.03 OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

1          DATA DAILY;
2             INPUT DATE MMDDYY8. TEMP SUNHRS;
3          CARDS;

NOTE: DATA SET WORK.DAILY HAS 1 OBSERVATIONS AND 4 VARIABLES. 529 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.08 SECONDS AND 336K.
      THE COMPILE   PHASE USED 0.05 SECONDS.
      THE EXECUTION PHASE USED 0.03 SECONDS.

14         PROC APPEND BASE=ACCUM.WEATHER FORCE;
NOTE: THESE VARIABLES ARE IN WORK.DAILY BUT NOT IN ACCUM.WEATHER: PRECIP.
NOTE: FORCE IS SPECIFIED, SO DROPPING/TRUNCATING WILL OCCUR.
NOTE: ACCUM.WEATHER HAS 4 OBSERVATION(S) BEFORE APPENDING.
NOTE: 1 OBSERVATION(S) ADDED.
NOTE: THE PROCEDURE APPEND USED 0.14 SECONDS AND 460K.
      THE COMPILE   PHASE USED 0.10 SECONDS.
      THE EXECUTION PHASE USED 0.04 SECONDS.

15         PROC PRINT;
16            FORMAT DATE DATE7.;
17            TITLE 'UP-TO-DATE WEATHER INFORMATION';
18            RUN;
NOTE: THE PROCEDURE PRINT USED 0.15 SECONDS AND 476K AND PRINTED PAGE 1.
      THE COMPILE   PHASE USED 0.11 SECONDS.
      THE EXECUTION PHASE USED 0.04 SECONDS.

19         RUN;
NOTE: SAS USED 476K MEMORY.
NOTE: THE COMPILE   PHASE USED 0.34 SECONDS.
      THE EXECUTION PHASE USED 0.14 SECONDS.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

```
                                  UP-TO-DATE WEATHER INFORMATION                1

                                 OBS      DATE       TEMP    SUNHRS

                                  1     01JAN84       35      9.30
                                  2     02JAN84       34      9.33
                                  3     03JAN84       37      9.35
                                  4     04JAN84       38      9.39
                                  5     05JAN84       35      9.40
```

Chapter 24

The BMDP Procedure

Operating systems: CMS and OS

ABSTRACT

The BMDP procedure calls any BMDP program to analyze data in a SAS data set.[1]

INTRODUCTION

BMDP® is a library of statistical analysis programs originally developed at the UCLA Health Sciences Computing Facility.

Use the BMDP procedure in SAS programs to:

- call a BMDP program to analyze data from a SAS data set
- convert a SAS data set to a BMDP save file.

PROC BMDP handles most versions of the P-series BMDP programs up to and including the 1983 version. It does not handle the older BMDP non-P programs. If your version of BMDP is older than the 1977 version, you must specify the UNIT= option in the PROC BMDP statement so that the SAS System can convert the data to a BMDP save file. For versions after 1977, the BMDP program can get data directly from a SAS data set.

To use PROC BMDP, first specify the name of the BMDP program that you want to invoke in the PROC BMDP statement;[2] follow this statement with the PARMCARDS statement and your BMDP control language. The SAS System prints the BMDP program output after the SAS log and any procedure output.

The BMDP Statistical Software Package is a registered trademark of BMDP Statistical Software Inc. The BMDP software is available from: BMDP Statistical Software Inc., 1964 Westwood Boulevard, Suite 202, Los Angeles, CA 90025.

You can analyze SAS data sets with BMDP programs by invoking this procedure. To analyze BMDP data with the SAS System, create a BMDP save file in a BMDP program, and then convert the save file to a SAS data set with the SAS CONVERT procedure. You can use pROC BMDP nay number of times in a SAS job to invoke BMDP.

SPECIFICATIONS

The PROC BMDP statement specifies which BMDP program to call. The VAR and BY statements can follow, but are optional. The BMDP control statements follow the PARMCARDS statement.

PROC BMDP *options*;
VAR *variables*;
BY *variables*;
PARMCARDS;
BMDP control statements
;

PROC BMDP Statement

PROC BMDP *options*;

The options listed below can be used in the PROC BMDP statement:

DATA=*SASdataset*
specifies the SAS data set that you want the BMDP program to process. If you do not specify the DATA= option, PROC BMDP uses the most recently created SAS data set.
Operating systems: CMS and OS

PROG=BMDP*nn*
specifies the BMDP program that you want to run. For example, the PROC BMDP statement

```
PROC BMDP PROG=BMDP3S;
```

runs the *BMDP3S* program.
Note: if you only want to convert a SAS data set to a BMDP save file and do not want to run a BMDP program, omit the PROG= option and include the UNIT= option, described below.
Operating systems: CMS and OS

UNIT=*n*
specifies the FORTRAN logical unit number for the BMDP save file that PROC BMDP creates. You must include UNIT= when you are

- using a version of BMDP released before 1977
- creating a BMDP save file (see the PROG= option above).

When you invoke SASBMDP, the UNIT= is automatically set to 3.[3] FT03F001 is the corresponding fileref for the save file. FT04F001 is the fileref for output save files. Do not specify FT01F001 or FT02F001 because BMDP reserves them for other uses. Add the appropriate control language to your job if you use any other unit.[4]
Operating systems: CMS and OS

CODE=*savefile*
assigns a name to the BMDP save file that PROC BMDP creates from a SAS data set. The *savefile* corresponds to the CODE sentence in the BMDP INPUT paragraph. For example, if you use the statement:

```
PROC BMDP PROG=BMDP3S CODE=JUDGES;
```

the BMDP INPUT paragraph must contain the sentence:

```
CODE='JUDGES'
```

CODE= is usually not necessary in the PROC BMDP statement. When CODE= is not specified, the name of the BMDP save file is the SAS data set name.

If you are converting a SAS data set to a BMDP save file, include the CODE sentence in the BMDP INPUT paragraph to name the save file. To use the name of the SAS data set, specify that name in the BMDP INPUT paragraph. If you use a different name, it must match the name supplied in the CODE= option.
Operating systems: CMS and OS

CONTENT= DATA
CONTENT=CORR
CONTENT=MEAN
CONTENT=FREQ

lets BMDP know if your SAS data set is a standard SAS data set (CONTENT=DATA) or if it contains a correlation matrix (CORR), variable means (MEAN), or frequency counts (FREQ). You need not specify the CONTENT= option for specially structured SAS data sets created by other SAS procedures if you omit the CONTENT= option, the data set's TYPE value is used.

Note: BMDP may use a structure for special data sets (for example, a correlation matrix) that is different from the SAS structure. Be sure that the input SAS data set is in the form that BMDP expects.
Operating systems: CMS and OS

LABEL=*variable*

specifies a variable whose values are to be used as case labels for BMDP. Only the first four characters of the values are used. The LABEL= variable must also be included in the VAR statement if you use one.
Operating systems: CMS and OS

LABEL2=*variable*

specifies a variable whose values are to be used as second case labels for BMDP; as with LABEL=, only the first four characters are used. The LABEL2= variable must also be given in the VAR statement if you use one.
Operating systems: CMS and OS

NOMISS

specifies that you want the BMDP program or save file to exclude observations with missing values.
Operating systems: CMS and OS

WRKSPCE=*nn*
PARM=*nn*

controls the allocation of a workspace in BMDP. The WRKSPCE= or PARM= value is passed as a parameter to BMDP programs and corresponds to the WRKSPCE= feature in BMDP OS cataloged procedures. The default is PARM=30.
Operating system: OS

VAR Statement

VAR *variables*;

The VAR statement specifies the variables to be used in the BMDP program. When you do not include a VAR statement, the BMDP program uses all the numeric variables in the SAS data set.

BY Statement

BY *variables*;

Use the BY statement with PROC BMDP to obtain separate analyses on observations in groups defined with the BY variables. When you use a BY statement, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

When you use a BY statement, the BY group information goes into the BMDP file's *LABEL IS* field.

PARMCARDS Statement

PARMCARDS;

The PARMCARDS statement signals that the BMDP control language follows.

Put your BMDP control language after the PARMCARDS statement. These are similar for all BMDP programs; see the most current BMDP manual for information on their forms and functions.

The BMDP INPUT paragraph must include UNIT and CODE sentences. Their values must match the UNIT= and CODE= values given in the PROC BMDP statement. (If the PROC BMDP statement does not specify a UNIT= value, use 3 as the UNIT value in the BMDP statements.) Use the SAS data set name as the CODE value unless you have specified a different name with the CODE= option in the PROC statement. Omit the VARIABLES paragraph from the BMDP statements since it is not needed when your input is a save file.

DETAILS

Missing Values

Before PROC BMDP sends data to BMDP, it converts missing SAS values to the BMDP missing value of 16^{32}. When you use the NOMISS option in the PROC BMDP statement, observations with missing values are excluded from the data set sent to the BMDP program.

Invoking BMDP Programs That Need FORTRAN Routines

Some BMDP programs, such as the ones for nonlinear regression, need to invoke the FORTRAN compiler and linkage editor before executing. All BMDP compilation and link editing must be completed before using PROC BMDP.

EXAMPLE

Creating and Converting a BMDP Save File

In the example below:

- DATA TEMP creates a SAS data set called TEMP.
- PROC CONTENTS shows the description information for the TEMP data set.
- PROC BMDP calls the BMDP program BMDP1D to analyze the TEMP data set.

  Note the BMDP program statements UNIT=3 and CODE='TEMP'. The results are stored in the BMDP save file, BOUT.

  Note that the word NEW must be in the SAVE paragraph. UNIT=*nn* should refer to the FTNNF001 fileref defined in your file definition statement.
- PROC CONVERT converts the BMDP save file BOUT to a SAS data set named FROMBMDP. BOUT is on UNIT 4, that is, FT04001.
- PROC CONTENTS and PROC PRINT show the new SAS data set, FROMBMDP.

```
DATA TEMP;
   INPUT A B C;
   CARDS;
1 2 3
4 5 6
7 8 9
PROC CONTENTS;
   TITLE 'CONTENTS OF SAS DATA SET TO BE RUN THROUGH BMDP1D';

PROC BMDP PROG=BMDP1D DATA=TEMP;
PARMCARDS;
/PROB TITLE='SHOW SAS/BMDP INTERFACE'.
/INPUT UNIT=3. CODE='TEMP'.
/SAVE CODE='BOUT'. NEW. UNIT=4.
/END
/FINISH
;

PROC CONVERT BMDP=FT04F001 OUT=FROMBMDP;
PROC CONTENTS;
   TITLE 'SAS DATA SET CONVERTED FROM BMDP SAVE FILE';
PROC PRINT;
```

Output 24.1 BOUT Save File Created from TEMP Data Set and Converted to SAS Data Set FROMBMDP

```
                          CONTENTS OF SAS DATA SET TO BE RUN THROUGH BMDP1D                    1

                                  CONTENTS OF SAS DATA SET WORK.TEMP

TRACKS USED=2  SUBEXTENTS=1  OBSERVATIONS=3  CREATED BY OS JOB BMDPEXM1  ON CPUID 11-3083-221547

AT 16:59 WEDNESDAY, OCTOBER 3, 1984     BY SAS RELEASE 5.03  DSNAME=SYS84277.T165945.RA000.BMDPEXM1.R0000001

BLKSIZE=19044  LRECL=28  OBSERVATIONS PER TRACK=680  GENERATED BY DATA

                                  ALPHABETIC LIST OF VARIABLES

          #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL

          1  A         NUM       8        4
          2  B         NUM       8       12
          3  C         NUM       8       20


                +------------------------------ SOURCE STATEMENTS ------------------------------+
                |   DATA TEMP;                                                                  |
                |      INPUT A B C;                                                             |
                |      CARDS;                                                                   |
                +-------------------------------------------------------------------------------+
```

```
                          SAS DATA SET CONVERTED FROM BMDP SAVE FILE                           2

                                CONTENTS OF SAS DATA SET WORK.FROMBMDP

TRACKS USED=2  SUBEXTENTS=1  OBSERVATIONS=3  CREATED BY OS JOB BMDPEXM1  ON CPUID XX-XXXX-XXXXXX

AT 16:59 WEDNESDAY, OCTOBER 3, 1984     BY SAS RELEASE 5.03  DSNAME=SYS12345.ABCDEFG.HIJKL.BMDPEXM1.MNOPQRST

BLKSIZE=19064  LRECL=20  TYPE=DATA  LABEL=   OCTOBER  3, 1984      17:00:09  OBSERVATIONS PER TRACK=953

GENERATED BY PROC CONVERT

                                  ALPHABETIC LIST OF VARIABLES

          #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL

          1  A         NUM       4        4
          2  B         NUM       4        8
          3  C         NUM       4       12
          4  USE       NUM       4       16
```

```
                          SAS DATA SET CONVERTED FROM BMDP SAVE FILE                           3

                                    OBS    A    B    C    USE

                                     1     1    2    3     1
                                     2     4    5    6     1
                                     3     7    8    9     1
```

```
PAGE   1

BMDP1D - SIMPLE DATA DESCRIPTION AND DATA MANAGEMENT
DEPARTMENT OF BIOMATHEMATICS
UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90024
(213) 825-5940          TWX  UCLA LSA
PROGRAM REVISED JUNE 1981
MANUAL REVISED -- 1981
COPYRIGHT (C) 1981 REGENTS OF UNIVERSITY OF CALIFORNIA
 OCTOBER  3, 1984  AT 17:00:09

TO SEE REMARKS AND A SUMMARY OF NEW FEATURES FOR
THIS PROGRAM, STATE NEWS. IN THE PRINT PARAGRAPH.

PROGRAM CONTROL INFORMATION

   /PROB TITLE='SHOW SAS/BMDP INTERFACE'.
   /INPUT UNIT=3. CODE='TEMP'.
   /SAVE CODE='BOUT'. NEW. UNIT=4.
   /END

PROBLEM TITLE IS
SHOW SAS/BMDP INTERFACE

NUMBER OF VARIABLES TO READ IN. . . . . .       3
NUMBER OF VARIABLES ADDED BY TRANSFORMATIONS. .       0
TOTAL NUMBER OF VARIABLES . . . . . .       3
NUMBER OF CASES TO READ IN. . . . . . TO END
CASE LABELING VARIABLES . . . . . .
MISSING VALUES CHECKED BEFORE OR AFTER TRANS. . NEITHER
BLANKS ARE. . . . . . MISSING
INPUT UNIT NUMBER . . . . . .       3
REWIND INPUT UNIT PRIOR TO READING. . DATA. . .     YES
NUMBER OF WORDS OF DYNAMIC STORAGE. . . . . .   59902

INPUT BMDP FILE
CODE. . . IS     TEMP
CONTENT . IS     DATA
LABEL . . IS
STATISTICAL ANALYSIS SYSTEM
VARIABLES
     1 A           2 B           3 C

VARIABLES TO BE USED
     1 A           2 B           3 C

-----------------------------------------
BMDP FILE IS BEING WRITTEN ON UNIT       4
CODE. . . IS     BOUT
CONTENT . IS     DATA
LABEL . . IS
  OCTOBER  3, 1984      17:00:09
```

```
PAGE   2     SHOW SAS/BMDP INTERFACE

VARIABLES ARE
     1 A           2 B           3 C


BMDP   FILE ON UNIT  4 HAS BEEN COMPLETED.
-----------------------------------------
NUMBER OF CASES WRITTEN TO FILE        3

NUMBER OF CASES READ. . . . . .       3
```

```
PAGE   3     SHOW SAS/BMDP INTERFACE

 VARIABLE      TOTAL                STANDARD    ST.ERR   COEFF. OF    S M A L L E S T     L A R G E S T
 NO. NAME      FREQUENCY      MEAN  DEVIATION   OF MEAN  VARIATION    VALUE   Z-SCORE    VALUE   Z-SCORE    RANGE

   1 A             3          4.000     3.000    1.7320   0.75000     1.000    -1.00      7.000    1.00      6.000
   2 B             3          5.000     3.000    1.7320   0.60000     2.000    -1.00      8.000    1.00      6.000
   3 C             3          6.000     3.000    1.7320   0.50000     3.000    -1.00      9.000    1.00      6.000

NUMBER OF INTEGER WORDS OF STORAGE USED IN PRECEDING    PROBLEM     184
CPU TIME USED      0.139 SECONDS
```

```
PAGE   4

BMDP1D - SIMPLE DATA DESCRIPTION AND DATA MANAGEMENT
 OCTOBER  3, 1984  AT 17:00:13

PROGRAM CONTROL INFORMATION

   /FINISH

NO MORE CONTROL LANGUAGE.

PROGRAM TERMINATED
```

REFERENCE

Dixon, W.D., Brown, M.B., Engleman, L., Frane, J.W., Hill, M.A., Jennrich, R.I., and Toporek, J.D., editors (1981), *BMDP Statistical Software 1981*, Los Angeles: University of California Press.

NOTES

1. **CMS**: PROC BMDP and the BMDP program it invokes require a large amount of memory. To be sure that you have sufficient memory, use the procedure in a 2-meg machine by entering these commands:

```
DEF STOR 2048K    (CP is entered after this command)
IPL CMS           (use to reinvoke CMS)
```

2. **OS**: The EXEC job control statement must request the cataloged procedure SASBMDP, rather than the usual cataloged procedure SAS.

If the SASBMDP cataloged procedure is not available on your computer or if it has a different name, ask your computing center staff for help in setting it up. Your site's SAS representative has the SAS System installation instructions that give directions.

CMS: Use the SASBMDP EXEC to invoke PROC BMDP.

If the SASBMDP EXEC is not available on your computer or if it has a different name, ask your computing center staff for help in setting it up. Your site's SAS representative has the SAS System installation instructions that give directions.

3. **CMS**: The SAS System has two BMDP interfaces. The older interface is invoked whenever the UNIT= option is specified.

PROC BMDP must operate on a MODULE file to run when the UNIT= option is **not** specified. BMDP Statistical Software distributes CMS programs in TEXT file form. If your BMDP programs are stored as TEXT files, you can:

- use the UNIT= option in the PROC BMDP statement
- request that some or all of the BMDP programs be made into MODULE files
- create MODULE files yourself and store them on your mini-disk.

For example, to create BMDP1D as a MODULE file, enter the following commands after accessing the disk containing the BMDP TEXT and the disk containing FORTRAN libraries:

```
GLOBAL TXTLIB BMDPFORT BMDPSUB FORTLIB
LOAD BMDP1D
GENMOD BMDP1D
```

PROC BMDP passes control to the BMDP program, which writes printed output to unit 6, wherever it is defined. If unit 6 is not defined in the control language, BMDP writes output to FILE FT06F001. If you use a SAVE paragraph in your BMDP

statements, BMDP writes a BMDP save file to the specified unit. For example, this FILEDEF command:

```
FILEDEF 4 DISK BMDPSAVE FILE (PERM
```

used with this SAVE paragraph:

```
/SAVE UNIT IS 4. NEW. NAME IS XXX.
```

creates a BMDP save file named XXX on unit 4 and stores the save file in the CMS file, BMDPSAVE FILE. (Note: not all BMDP programs can create output save files.) To perform additional SAS analysis on the output save file, create a SAS data set with PROC CONVERT. For example, the following statement:

```
PROC CONVERT BMDP=FT04F001 OUT=FROMBMDP;
```

creates the SAS data set, FROMBMDP, from the BMDP save file, XXX.

4. **OS**:

```
//FTnnF001 DD UNIT=SYSDA,
//    SPACE=(3520,(100,20)),
//    DCB=(RECFM=VBS,BLKSIZE=3520,
//    LRECL=3515)
```

TSO:

```
ATTR ATR1 RECFM(V B S) LRECL(3515) BLKSIZE(3520)
ALLOC F(FTnnF001) SPACE(3520 100) BLOCKS USING(ATR1)
```

CMS:

```
FILEDEF nn DSK filename filetype mode
   (RECFM VBS LRECL 3515 BLKSIZE 3520)
```

where *nn* is the unit you use.

Chapter 25

The BROWSE Procedure

Operating systems: All

ABSTRACT

The BROWSE procedure enables you to read a SAS data set. PROC BROWSE is primarily an interactive procedure, but it can be used in batch mode.

INTRODUCTION

The BROWSE procedure is the same as the EDITOR procedure except that certain commands used to modify the data set in PROC EDITOR are not allowed: REPLACE, ADD, DELETE, and DUP. Since you are not altering the data set, you can read a SAS data set to which you have only read access.

SPECIFICATIONS

These statements and commands can be used with PROC BROWSE:

PROC BROWSE *options*;
  FORMAT *variable format. ...*;
  INFORMAT *variable format. ...*;
  FIND *options range variable1 operator1value1 ...*;
  LOCATE *options range value ...*;
  SEARCH *options range string ...*;
  NAME *variable*;
  STRING *variable ...*;
  VERIFY *option*;
  END;
  LIST *range variable ...*;
  TOP;
  BOTTOM;
  UP *n*;
  DOWN *n*;

Refer to PROC EDITOR for details on using PROC BROWSE. Everything described in PROC EDITOR applies to PROC BROWSE except documentation on the change commands: REPLACE, DELETE, DUP, and ADD.

Chapter 26

The CALENDAR Procedure

Operating systems: All

ABSTRACT

The CALENDAR procedure displays data from a SAS data set in a month-by-month calendar format.

INTRODUCTION

PROC CALENDAR produces either of two calendar formats: summary or schedule.

A summary calendar provides information about the same variables over time. For example, the summary calendar below displays the number of meals served daily in a hospital cafeteria during the month of December, 1985.

Output 26.1 Summary Calendar

MEALS SERVED IN COMMUNITY HOSPITAL CAFETERIA 1

DECEMBER 1985

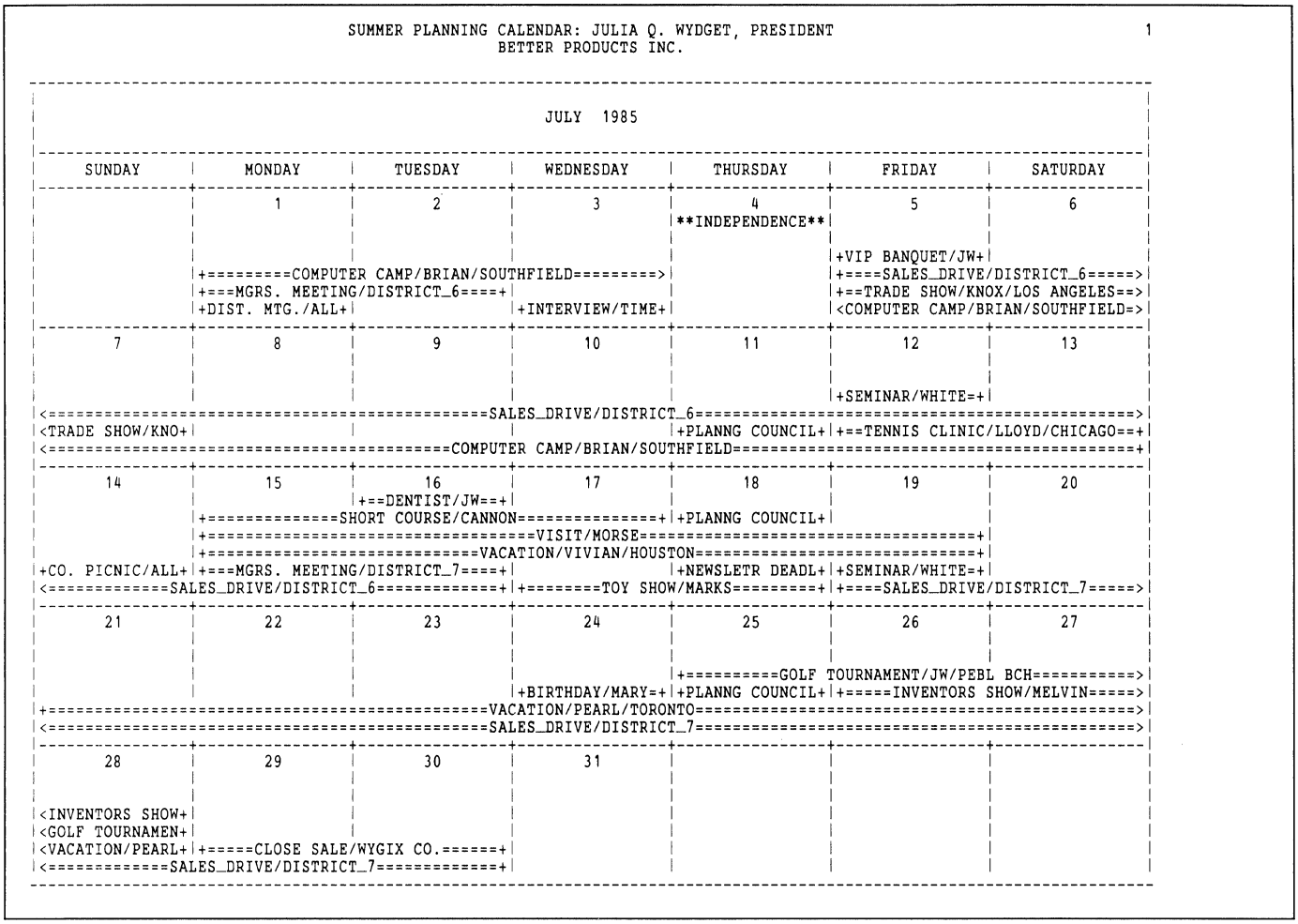
SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
1	2	3	4	5	6	7
334 BRKFST 484 LUNCH 368 DINNER	289 BRKFST 501 LUNCH 380 DINNER	184 BRKFST 352 LUNCH 292 DINNER	179 BRKFST 228 LUNCH 212 DINNER	360 BRKFST 450 LUNCH 332 DINNER	245 BRKFST 440 LUNCH 386 DINNER	280 BRKFST 399 LUNCH 312 DINNER
8	9	10	11	12	13	14
294 BRKFST 489 LUNCH 316 DINNER	309 BRKFST 569 LUNCH 449 DINNER	199 BRKFST . 202 DINNER	186 BRKFST 294 LUNCH 260 DINNER	301 BRKFST 511 LUNCH 403 DINNER	276 BRKFST 488 LUNCH 365 DINNER	272 BRKFST 460 LUNCH 421 DINNER
15	16	17	18	19	20	21
204 BRKFST 356 LUNCH 345 DINNER	266 BRKFST 480 LUNCH 470 DINNER	195 BRKFST 388 LUNCH 311 DINNER	164 BRKFST 280 LUNCH 234 DINNER	210 BRKFST 394 LUNCH 256 DINNER	311 BRKFST 595 LUNCH 541 DINNER	402 BRKFST 654 LUNCH 580 DINNER
22	23	24	25 *CHRISTMAS**	26	27	28
420 BRKFST 530 LUNCH 501 DINNER	339 BRKFST 590 LUNCH 489 DINNER	201 BRKFST 400 LUNCH 266 DINNER	160 BRKFST 485 LUNCH 195 DINNER	412 BRKFST 663 LUNCH 581 DINNER	292 BRKFST 559 LUNCH 503 DINNER	309 BRKFST . 552 DINNER
29	30	31				
330 BRKFST 771 LUNCH 495 DINNER	321 BRKFST 605 LUNCH 550 DINNER	220 BRKFST 415 LUNCH .				

LEGEND	SUM	MEAN
BREAKFASTS SERVED	8464	273.032
LUNCHES SERVED	13830	476.897
DINNERS SERVED	11567	385.567

In a summary calendar, each day is represented by one observation. Each piece of information for a given day is the value of a variable for that day. Variables displayed can be numeric or character and can be formatted. Sums and means for any of the numeric variables displayed can be collected; these are printed after the body of the calendar.

A schedule calendar displays the duration of an event by a continuous line through each day for the event. The example below is a planning calendar for the month of July, 1985.

Output 26.2 Schedule Calendar



```
                    SUMMER PLANNING CALENDAR: JULIA Q. WYDGET, PRESIDENT                    1
                                    BETTER PRODUCTS INC.

                                                JULY  1985

     SUNDAY     |     MONDAY     |    TUESDAY     |   WEDNESDAY    |    THURSDAY    |     FRIDAY     |    SATURDAY
                |        1       |        2       |        3       |        4       |        5       |        6
                |                |                |                |**INDEPENDENCE**|                |
                |                |                |                |                |+VIP BANQUET/JW+|
                |+=========COMPUTER CAMP/BRIAN/SOUTHFIELD=========>|                |+====SALES_DRIVE/DISTRICT_6=====>|
                |+===MGRS. MEETING/DISTRICT_6====+|                |                |+==TRADE SHOW/KNOX/LOS ANGELES==>|
                |+DIST. MTG./ALL+|                |+INTERVIEW/TIME+|                |<COMPUTER CAMP/BRIAN/SOUTHFIELD=>|
       7        |        8       |        9       |       10       |       11       |       12       |       13
                |                |                |                |                |                |
                |                |                |                |                |+SEMINAR/WHITE=+|
<===============================================SALES_DRIVE/DISTRICT_6===============================================>|
<TRADE SHOW/KNO+|                |                |                |+PLANNG COUNCIL+|+==TENNIS CLINIC/LLOYD/CHICAGO==+|
<===============================================COMPUTER CAMP/BRIAN/SOUTHFIELD=================================================+|
       14       |       15       |       16       |       17       |       18       |       19       |       20
                |                |+==DENTIST/JW==+|                |                |                |
                |+==============SHORT COURSE/CANNON===============+|+PLANNG COUNCIL+|                |
                |+=================================VISIT/MORSE=================================+|                |
                |+============================VACATION/VIVIAN/HOUSTON============================+|                |
+CO. PICNIC/ALL+|+===MGRS. MEETING/DISTRICT_7====+|                |+NEWSLETR DEADL+|+SEMINAR/WHITE=+|                |
<==============SALES_DRIVE/DISTRICT_6==============+|+========TOY SHOW/MARKS==========+|+====SALES_DRIVE/DISTRICT_7=====>|
       21       |       22       |       23       |       24       |       25       |       26       |       27
                |                |                |                |                |                |
                |                |                |                |+==========GOLF TOURNAMENT/JW/PEBL BCH==========>|
                |                |                |+BIRTHDAY/MARY=+|+PLANNG COUNCIL+|+=====INVENTORS SHOW/MELVIN=====>|
+=============================================VACATION/PEARL/TORONTO=============================================>|
<===============================================SALES_DRIVE/DISTRICT_7===============================================>|
       28       |       29       |       30       |       31       |                |                |
                |                |                |                |                |                |
<INVENTORS SHOW+|                |                |                |                |                |
<GOLF TOURNAMEN+|                |                |                |                |                |
<VACATION/PEARL+|+=====CLOSE SALE/WYGIX CO.======+|                |                |                |
<==============SALES_DRIVE/DISTRICT_7==============+|                |                |                |
```

The line for an event begins and ends with a cross (+). If an event continues from one week to another or after one or more holidays, an arrow appears at the point of continuation. In a schedule calendar, each event is one observation, and a single day can display several events (observations) on successive lines. One event can also appear in several days, weeks, or months, depending on its duration.

PROC CALENDAR determines the amount of data that can be displayed with either calendar type from the number of lines available on the page (PAGESIZE= system option) and the line width (LINESIZE= or TLINESIZE= system option). See "SAS System Options" for more information on these options.

SPECIFICATIONS

The CALENDAR procedure is invoked by using the following statements:

PROC CALENDAR *options*;
  BY *variables*;
  ID *variable*;
  VAR *variables*;

SUM *variables / option*;
MEAN *variables / option*;
DURATION *variable*;
HOLIDAYS *variables*;
HOLINAME *variable*;

PROC CALENDAR Statement

PROC CALENDAR *options*;

The following options can appear on the PROC CALENDAR statement and can be used with either a summary or a schedule calendar:

DATA=*SASdataset*
: gives the name of the SAS data set to be used by the CALENDAR procedure. If DATA= is omitted, the most recently created SAS data set is used.

HOLIDATA=*SASdataset*
: gives the name of the SAS data set that contains variables whose values are "holidays." The variables which contain holiday values are specified in the HOLIDAYS statement. See **HOLIDATA= Data Set** under **Details** for information on the HOLIDATA= data set.

SCHEDULE
: specifies that a schedule calendar is to be printed. A summary calendar is the default if no DURATION statement is specified.

FILL
: specifies that all months between the first and last observation dates inclusive are to be displayed even if no data are present for a particular month. By default, CALENDAR leaves out months with no data.

WEEKDAYS
: specifies that CALENDAR should display and use in calculations only weekdays (excluding Saturday and Sunday). The default displays and uses all days.

MISSING
: specifies for a summary calendar that when a day has no associated observation, the values of the VAR variables for that day are to be printed using the format specified for missing values. If MISSING is not given, the day contains only the date. For a schedule calendar, MISSING specifies that missing values for VAR variables appear in the label of an observation. If MISSING is not specified, missing values of VAR variables are ignored in labeling the observation.

DATETIME
: specifies that the ID and HOLIDAYS variables contain SAS datetime values rather than SAS date values. PROC CALENDAR then uses only the date portion of the datetime value. Use this option if the ID and HOLIDAYS variables are associated with the DATETIME., SMFSTAMP., or RMFSTAMP. informats. Both the ID and HOLIDAYS variables must contain the same type of value. By default, CALENDAR assumes that the ID and HOLIDAYS variables are SAS date values.

HEADER=LARGE
HEADER=SMALL
: specifies the type of heading for CALENDAR to use in formatting the month name. HEADER=LARGE prints the month name 7 lines high

with an additional blank line above and below the month name. The year is also printed if space is available (as determined by the LINESIZE= option). HEADER=SMALL prints the month and year on a single line like a regular printed title. By default, CALENDAR prints the month and year in a box 4 lines high.

FORMCHAR='*string*'
FORMCHAR(*n* ...)='*string*'
defines the set of characters used to draw the calendar outlines and the special characters used to represent events on a schedule calendar. If the FORMCHAR(*n* ...)='*string*' form of the option is used, each *n* must represent a valid position in the FORMCHAR= string. See **Using the FORMCHAR= Option** below.

The following options can be used for a summary calendar (they are ignored if the SCHEDULE option is specified):

LEGEND
requests that CALENDAR print a legend block containing the label or name of each variable displayed on the calendar. The legend appears at the bottom of the page for each month or on a following page if there is insufficient room on the calendar page.

MEANTYPE=NOBS
MEANTYPE=NDAYS
gives the type of mean to be calculated for each month. MEANTYPE=NOBS specifies that the mean is to be calculated for the number of observations on the input data set for each month. MEANTYPE=NDAYS specifies that the mean is to be calculated for the number of days displayed in the month (the number of days displayed depends on the WEEKDAYS option). The default MEANTYPE is NOBS.

BY Statement

BY *variables*;

A BY statement can be used with PROC CALENDAR to obtain separate calendars for observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables, in addition to the ID variable. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

ID Statement

ID *variable*;

The ID statement is required, and the ID variable must contain either SAS date values or SAS datetime values. If the values are datetime values, specify the DATETIME option in the PROC CALENDAR statement. The input data set must be sorted by the ID variable (within any BY variables). If a HOLIDAYS statement is present, the ID variable values and the HOLIDAYS variable values must contain the same type of information (either date or datetime).

For a summary calendar, the ID statement specifies the variable to be used as the date of each observation. There should be only one observation per day in

the input data set; if multiple observations for the same day appear, only the last is used. Observations with missing values for the ID variable are ignored.

For a schedule calendar, the ID statement specifies the variable used as the beginning date for the observation (event). Several observations may have the same ID value.

VAR Statement

VAR *variables*;

For a summary calendar, the VAR statement specifies the variables in the input data set to be displayed on the calendar for each day. Variables can be either character or numeric. If no VAR statement is used, all variables in the data set (excluding BY and ID variables) are displayed. Note that the number of lines available per page determines the actual number of variables that a summary calendar displays.

For a schedule calendar, the values of the VAR variables label each observation and are printed in the middle of the line for a given observation. VAR variables can be character or numeric. The number of VAR variable values that can be printed for a given observation depends on the number of days that observation occupies, that is, the length of the observation's line.

SUM Statement

SUM *variables / option*;

The SUM statement specifies for a summary calendar the variables to be totaled for each month. If a variable specified in the SUM list is not in the VAR list, it is added to the VAR list. Multiple SUM statements can be used.

The SUM statement is valid only for a summary calendar; if the SCHEDULE option is specified, SUM statements are ignored.

The following option can be used with the SUM statement:

FORMAT=*formatname*
F=*formatname*
: names a SAS or user-defined format to be used in displaying the sums requested. The default uses the BEST. format for the sums.

MEAN Statement

MEAN *variables / option*;

The MEAN statement specifies for a summary calendar the variables for which a mean value is to be calculated for each month. If a variable specified in the MEAN list is not in the VAR list, SAS adds that variable to the VAR list. Multiple MEAN statements can be used.

The MEAN statement is valid only for a summary calendar; if the SCHEDULE option is specified, MEAN statements are ignored.

The following option can be used with the MEAN statement:

FORMAT=*formatname*
F=*formatname*
: names a SAS or user-defined format to be used in displaying the means requested. The default uses the BEST. format for the means.

DURATION Statement

DURATION *variable*;

The DURATION (or DUR) statement specifies for a schedule calendar the variable containing the duration of each event in days. The DUR statement is required for a schedule calendar, and using a DUR statement causes SAS to produce a schedule calendar.

Duration is measured inclusively from the first day of an event (given in the ID variable). For example, the line for an event with an ID value of 1JUL85 and a DUR value of 3 appears in July 1, July 2, and July 3 of a schedule calendar. Any non-integer portion of the DURATION variable value is truncated, and any observation with a missing value or a value less than or equal to 0 is skipped. If a HOLIDAYS statement is present, the DUR value does not include any holidays within the duration. Thus, if 4JUL85 is a holiday, an event with an ID value of 1JUL85 and a DUR value of 4 appears in July 1, 2, 3, and 5. If you use the WEEKDAYS option, CALENDAR assumes that DUR refers to the number of weekdays. Specifying the WEEKDAYS option causes an event with an ID value of 1JUL85 and a DUR value of 31 to extend into August.

HOLIDAYS Statement

HOLIDAYS *variables;*

The HOLIDAYS statement specifies a list of variables in the HOLIDATA= data set to be treated as "holidays." For a schedule calendar, a holiday is defined as a day to be excluded from an event. If an event spans a holiday, it is simply continued past the holiday. For a summary calendar, the holiday name is placed above the values for the holiday. If multiple observations or variables in the HOLIDATA= data set have the same value the last value encountered is used.

HOLINAME Statement

HOLINAME *variable;*

The HOLINAME statement specifies a variable in the HOLIDATA= data set whose formatted value in each observation labels all holidays specified in the HOLIDAYS statement for that observation. The variable can be either character or numeric. If no HOLINAME statement is used, the holiday is labeled with the variable label, if present, or the name of the HOLIDAYS variable that contributed the date.

DETAILS

Input Data Set

The data set must be sorted by the ID variable (within any BY variables).

For a summary calendar the input data set should consist of one observation per day. If multiple observations for one day are encountered, only the last is used. You can use PROC SUMMARY or a DATA step to collapse the data by day.

For a schedule calendar the input data set should contain one observation per event.

HOLIDATA= Data Set

The HOLIDATA= data set contains variables to identify and, optionally, label days to be considered holidays in the calendar. You can treat any day as a holiday.

Variables that identify holidays must contain either SAS date or SAS datetime values. In PROC CALENDAR, specify these variables in the HOLIDAYS statement. The data set does not have to be sorted by the HOLIDAYS variables.

Specify the variable that labels the holidays in the HOLINAME statement. The HOLINAME variable labels holidays for all variables in the HOLIDAYS statement for each observation. If the value of the HOLINAME variable is too long for a calendar cell, the value is truncated.

These statements create a HOLIDATA= data set:

```
DATA HOL;
   INPUT HDAY : DATE7. HNAME & $12.;
   CARDS;
1JAN85  NEW YEAR'S
4JUL85  INDEPENDENCE
2SEP85  LABOR DAY
25DEC85 CHRISTMAS
3APR85  CONVENTION
4APR85  CONVENTION
5APR85  CONVENTION
30JUN85 TAKEINVENTRY
;
PROC PRINT;
   ATTRIB HDAY FORMAT=DATE7.;
RUN;
```

Output 26.3 HOLIDATA= Data Set

HOLIDATA= DATA SET 1

OBS	HDAY	HNAME
1	01JAN85	NEW YEAR'S
2	04JUL85	INDEPENDENCE
3	02SEP85	LABOR DAY
4	25DEC85	CHRISTMAS
5	03APR85	CONVENTION
6	04APR85	CONVENTION
7	05APR85	CONVENTION
8	30JUN85	TAKEINVENTRY

Missing Values

Table 26.1 shows how CALENDAR handles missing values of various types.

Limitations

Since PROC CALENDAR attempts to fit the calendar within a single page, the page dimensions determine the number of variables and observations that can be displayed. You should carefully select the values for the system options PAGESIZE= and LINESIZE= or TLINESIZE=. CALENDAR prints a message on the SAS log if one or more events is omitted for lack of space on a schedule calendar or if one or more variables is omitted for a summary calendar.

Using the FORMCHAR= Option

PROC CALENDAR uses the FORMCHAR= values to produce the calendar outlines and the event representation lines on a schedule calendar. The boxing characters (characters 1 to 11 in the FORMCHAR= string) are used to generate the calendar outlines. The vertical thick character (character 12) is the starting or ending character for an event line. The horizontal thick character (character 13) is the fill character for the event line. The slash (character 16) separates the variable

values in the event line label. The left and right arrows (characters 18 and 19) indicate the continuation of an event line. The bullet (character 20) highlights a holiday name.

Table 26.1 Treatment of Missing Values in PROC CALENDAR

	Action	
Type of Missing Data	Summary	Schedule
No observation for day	MISSING option specified: Variables represented by missing values MISSING option not specified: day in calendar contains only date	
Missing value for ID variable	Observation ignored	Observation ignored
Missing value for DUR variable		Observation ignored
Missing value for HOLIDAYS variable	Value not treated as a holiday	Value not treated as a holiday
Missing value for HOLINAME variable	Day labeled with missing value	Day labeled with missing value
Missing value for VAR variable	Value represented by a missing value for that day; treatment of other VAR; variables for that day not affected	MISSING option specified: variables represented by missing values MISSING option not specified: missing value ignored in labeling event lines
Missing value for SUM variable	Value ignored in calculating sum	
Missing value for MEAN variable	Value included in calculating mean unless ID variable missing	

You can change the FORMCHAR= characters with the system option FORMCHAR= or the FORMCHAR= option in the PROC CALENDAR statement. Use FORMCHAR=*'string'* to change all the characters or a group of consecutive characters beginning with the first. Use FORMCHAR(*n . . .*)=*'string'* to change

only the characters at positions *n*. . . . For example, FORMCHAR(18 19)='XX' changes the left and right arrows to X.

The default value of the FORMCHAR= option is ` |----|+|---+=|-/\<>*'. If you use an IBM 6670 printer with an extended font (typestyle 27 or 225) with input character set 216, we recommend that you specify

```
FORMCHAR='FABFACCCBCEB8FECABCBBB4E7E4F6061E04C6EAF'X
```

If you use an IBM 1403, 3211 or 3203-5 printer, or equivalent, with a TN (text) print train, we recommend that you specify

```
FORMCHAR='4FBFACBFBC4F8F4FABBFBB4E7E4F6061E04C6EAF'X
```

Table 26.2 shows the characters produced by the default value and by the two suggested values. You can substitute any character or hexadecimal string for those given here to customize the appearance of the calendar.

Table 26.2 Characters Used with the FORMCHAR= Option

Name	Position	Default	TN(text) print train	IBM 6670 printer with extended font
Vertical bar	1	|	|	|
		4F	4F	FA
Horizontal bar	2	-	-	-
		60	BF	BF
Upper left	3	-	┌	┌
		60	AC	AC
Upper middle	4	-	-	┬
		60	BF	CC
Upper right	5	-	┐	┐
		60	BC	BC
Middle left	6	|	|	├
		4F	4F	EB
Middle middle	7	+	┼	┼
		4E	8F	8F
Middle right	8	|	|	┤
		4F	4F	EC

continued on next page

Table 26.2 *Continued*

Name	Position	Default	TN(text) print train	IBM 6670 printer with extended font
Lower left	9	- 60	└ AB	└ AB
Lower middle	10	- 60	- BF	┴ CB
Lower right	11	- 60	┘ BB	┘ BB
Vertical thick	12	+ 4E	+ 4E	+ 4E
Horizontal thick	13	= 7E	= 7E	= 7E
Vertical dash	14	| 4F	| 4F	| 4F
Horizontal dash	15	- 60	- 60	- 60
Slash	16	/ 61	/ 61	/ 61
Backslash	17	\ E0	\ E0	\ E0
Left arrow	18	< 4C	< 4C	< 4C
Right arrow	19	> 6E	> 6E	> 6E
Bullet	20	* 5C	• AF	• AF

Printed Output

The printed output for a summary calendar consists of the following:

1. any titles specified
2. the month name and year heading
3. day of the week heading
4. data for each day listed week by week
5. a combination legend, sums, and means block if these are requested.

The legend/summary block appears on a following page if it does not fit on the same page as the calendar.

The printed output for a schedule calendar consists of items 1 through 3 above and a week-by-week listing of the events on each day. Each event is represented by a continuous line through the days on which it occurs except for holidays. The values of the variables specified in the VAR list are used to label this line. An arrow at the right end of an event line indicates that the event continues into the next week or past the following holiday. An arrow at the left end of an event line indicates that the event continues from the previous day or week.

EXAMPLES

Summary Calendar: Example 1

The summary calendar displays computer usage at the Dusty National Bank for the month of March, 1983. Note that the calendar records only use from Monday through Friday of each week (the WEEKDAYS option). The mean is calculated on the number of days displayed in the month (MEANTYPE=NDAYS). Values for March 4 (for which there is no observation) are printed using the format for missing values (the MISSING option). This example uses the value of the FORMCHAR= option suggested for IBM 6670 printer with an extended font (typestyle 225).

```
OPTIONS LINESIZE=120 NODATE NONUMBER;
DATA CAL;
   INPUT OPDATE: DATE. JOBS ACT CPU;
   LIST;
   CARDS;
1MAR83 873 22.1 7.6
2MAR83 881 23.8 11.7
3MAR83 940 24.0 7.7
5MAR83 877 23.5 9.1
6MAR83 194 5.3 1.5
7MAR83 154 17.4 7.1
8MAR83 807 24 10.5
9MAR83 829 23.4 10.5
10MAR83 915 24 10.6
11MAR83 582 19.2 6.2
12MAR83 647 17.1 4.3
13MAR83 388 21 6.9
14MAR83 194 22.5 10.9
15MAR83 806 23.5 7.5
16MAR83 848 23.6 10.5
17MAR83 906 23 10.1
18MAR83 505 14.9 5.8
19MAR83 4 0.1 0
21MAR83 3 0.1 0
22MAR83 729 24 4.9
23MAR83 652 21.3 12.2
24MAR83 809 23.8 12.3
25MAR83 15 3.5 1.6
26MAR83 103 23.4 18.7
27MAR83 168 15.8 10.7
28MAR83 108 15.3 6
```

```
29MAR83 896 21.5 5.8
30MAR83 760 18.7 8
31MAR83 200 10 5
;
PROC FORMAT;
   PICTURE JFMT . ='   0 JOBS' (NOEDIT)
            OTHER=' 999 JOBS';
   PICTURE AFMT . ='   0 ACT ' (NOEDIT)
            OTHER='99. 9 ACT ';
   PICTURE CFMT . ='   0 CPU ' (NOEDIT)
            OTHER='99. 9 CPU ';
PROC CALENDAR WEEKDAYS HEADER=SMALL MEANTYPE=NDAYS MISSING
   FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   ID OPDATE;
   VAR JOBS ACT CPU;
   SUM JOBS ACT CPU;
   MEAN JOBS ACT CPU;
   FORMAT JOBS JFMT.
          ACT  AFMT.
          CPU  CFMT. ;
   LABEL JOBS = 'JOBS RUN DURING THE MONTH'
         ACT  = 'ACTIVE HOURS'
         CPU  = 'CPU HOURS';
   TITLE1 'SYSTEM PERFORMANCE SUMMARY';
   TITLE2 'DUSTY NATIONAL BANK COMPUTER CENTER';
```

Output 26.4 Summary Calendar

SYSTEM PERFORMANCE SUMMARY
DUSTY NATIONAL BANK COMPUTER CENTER

MARCH 1983

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
	1 873 JOBS 22. 0 ACT 01. 0 CPU	2 881 JOBS 23. 0 ACT 08. 0 CPU	3 940 JOBS 24. 0 ACT 00. 0 CPU	4 0 JOBS 0 ACT 0 CPU
7 154 JOBS 17. 0 ACT 04. 0 CPU	8 807 JOBS 24. 0 ACT 10. 0 CPU	9 829 JOBS 23. 0 ACT 04. 0 CPU	10 915 JOBS 24. 0 ACT 10. 0 CPU	11 582 JOBS 19. 0 ACT 02. 0 CPU
14 194 JOBS 22. 0 ACT 05. 0 CPU	15 806 JOBS 23. 0 ACT 05. 0 CPU	16 848 JOBS 23. 0 ACT 06. 0 CPU	17 906 JOBS 23. 0 ACT 10. 0 CPU	18 505 JOBS 14. 0 ACT 09. 0 CPU
21 003 JOBS 00. 0 ACT 01. 0 CPU	22 729 JOBS 24. 0 ACT 04. 0 CPU	23 652 JOBS 21. 0 ACT 03. 0 CPU	24 809 JOBS 23. 0 ACT 08. 0 CPU	25 015 JOBS 03. 0 ACT 05. 0 CPU
28 108 JOBS 15. 0 ACT 03. 0 CPU	29 896 JOBS 21. 0 ACT 05. 0 CPU	30 760 JOBS 18. 0 ACT 07. 0 CPU	31 200 JOBS 10. 0 ACT 05. 0 CPU	

	SUM	MEAN
JOBS RUN DURING THE MONTH	13412	583.13
ACTIVE HOURS	416	18.087
CPU HOURS	115	5

Schedule Calendar: Example 2

The following SAS statements produced the schedule calendar shown in the **Introduction**. The HOLIDATA= data set is shown in the **HOLIDATA= Data Set** section. Since the MISSING option is not specified, the missing values for WHERE are ignored in labeling the events.

```
DATA PLAN;
   INPUT DATE:DATE. HAPPEN $ 9-36 WHO $ 38-48 LONG 50-51 WHERE $ 53-70;
   CARDS;
1JUL85  DIST. MTG.                ALL          1  CARY
1JUL85  MGRS. MEETING             DISTRICT_6   2
1JUL85  COMPUTER CAMP             BRIAN        12 SOUTHFIELD
5JUL85  TRADE SHOW                KNOX         3  LOS ANGELES
12JUL85 TENNIS CLINIC             LLOYD        2  CHICAGO
16JUL85 DENTIST                   JW           1
18JUL85 NEWSLETR DEADLN           ALL          1
5JUL85  SALES_DRIVE               DISTRICT_6   12
11JUL85 PLANNG COUNCIL            GROUP II     1
15JUL85 MGRS. MEETING             DISTRICT_7   2
15JUL85 VACATION                  VIVIAN       5  HOUSTON
15JUL85 VISIT                     MORSE        5
```

```
19JUL85 SALES_DRIVE                DISTRICT_7  12
25JUL85 PLANNG COUNCIL             GROUP IV    1
5JUL85  VIP BANQUET                JW          1
15JUL85 SHORT COURSE               CANNON      3
3JUL85  INTERVIEW                  TIMES       1
29JUL85 CLOSE SALE                 WYGIX CO.   2
16JUL85 BANK MTG.                  1ST NATL    1
24JUL85 BIRTHDAY                   MARY        1
17JUL85 TOY SHOW                   MARKS       2
26JUL85 INVENTORS SHOW             MELVIN      3
14JUL85 CO. PICNIC                 ALL         1  PARK
19JUL85 SEMINAR                    WHITE       1
21JUL85 VACATION                   PEARL       8  TORONTO
25JUL85 GOLF TOURNAMENT            JW          4  PEBL BCH
12JUL85 SEMINAR                    WHITE       1
18JUL85 PLANNG COUNCIL             GROUP III   1
;
PROC SORT;
   BY DATE;
PROC CALENDAR SCHEDULE HOLIDATA=HOL;
   ID DATE;
   HOLIDAYS HDAY;
   HOLINAME HNAME;
   VAR HAPPEN WHO WHERE;
   DUR LONG;
TITLE 'SUMMER PLANNING CALENDAR: JULIA Q. WYDGET, PRESIDENT';
TITLE2 'BETTER PRODUCTS INC.';
```

Chapter 27

The CHART Procedure

Operating systems: All

ABSTRACT

The CHART procedure produces vertical and horizontal bar charts (also called *histograms*), block charts, pie charts, and star charts. These charts are useful for showing pictorially a variable's values or the relationships between two or more variables.

INTRODUCTION

Method

The appearance of a chart produced by PROC CHART is determined by three factors:

1. the method chosen to present the chart
2. the summary measures that are shown for the variable whose values are charted
3. features of the procedure that specify how the values are grouped.

You request each chart presentation method in PROC CHART with the following statements:

- vertical bar chart (VBAR statement)
- horizontal bar chart (HBAR statement)
- block chart (BLOCK statement)
- pie chart (PIE statement)
- star chart (STAR statement).

In each case, the variable listed (the chart variable) determines the values that label the bars or sections. Options are used to control the kind of statistics presented and any grouping that is done. For example, you use the TYPE= option to choose a measure to compute and display:

- frequency counts (TYPE=FREQ)
- percentages (TYPE=PCT)
- cumulative frequencies (TYPE=CFREQ)
- cumulative percentages (TYPE=CPCT)
- totals (TYPE=SUM)
- averages (TYPE=MEAN).

Among the characteristics that determine how values are grouped are

- the type of variable charted, numeric or character
- the DISCRETE option for categorical grouping variables
- the GROUP= option for side-by-side grouping
- the SUBGROUP= option for sub-grouping
- the MIDPOINTS= option to locate interval midpoints for continuous variables.

PROC CHART produces charts for both numeric and character variables. Character variables and formats cannot exceed a length of sixteen. For continuous numeric variables, CHART automatically selects display intervals, although you can explicitly define interval midpoints. If a value falls exactly halfway between two midpoints, the procedure puts the value in the higher interval. For character variables and discrete numeric variables, which contain several distinct values rather than a continuous range, the data values themselves define the intervals.

Introductory Examples

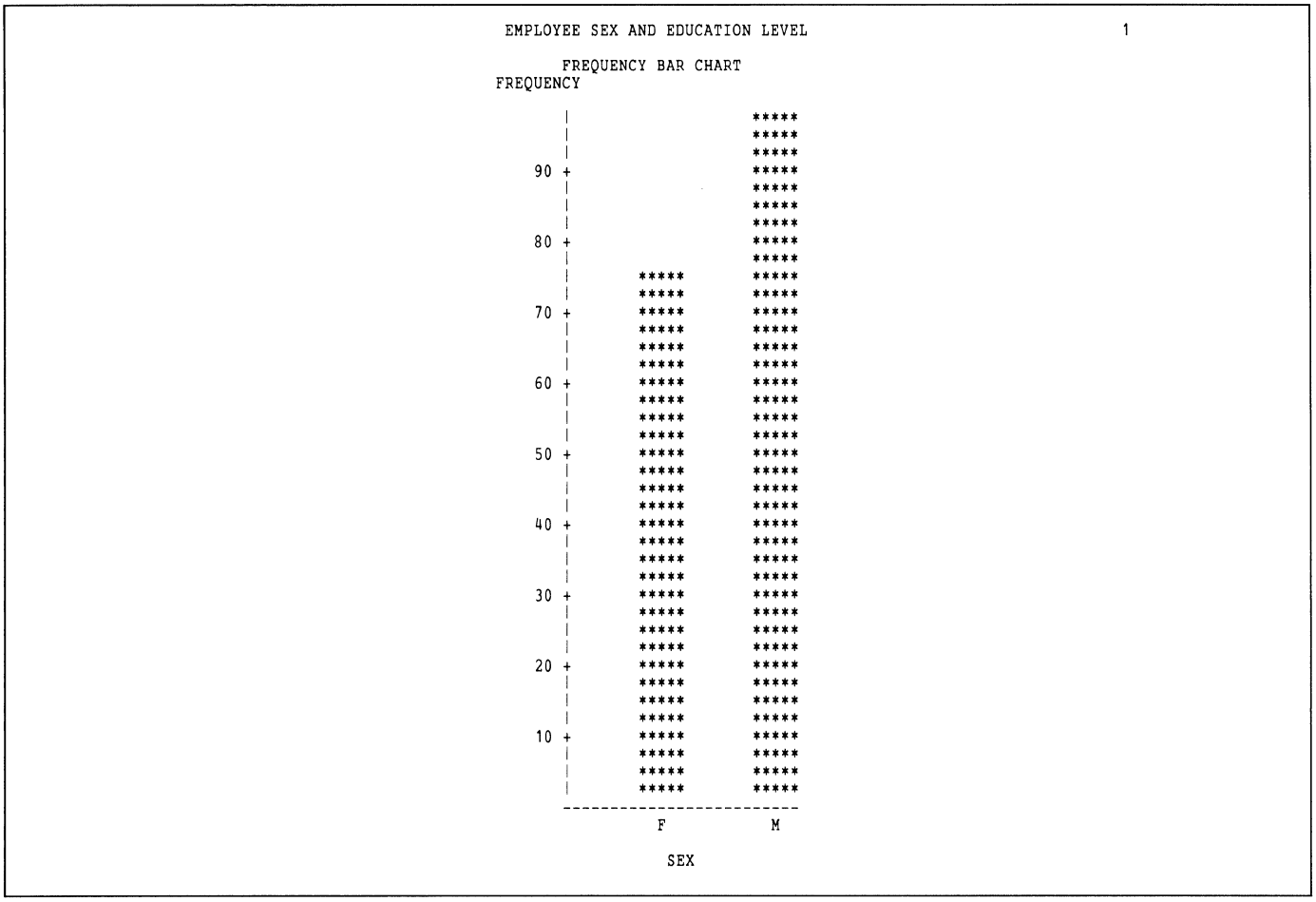
To give you an idea of PROC CHART's capabilities, the kinds of charts that you can produce are described below. If you have a specific chart in mind, glance through the following examples for a similar one. Many other charts can be produced by combining the options. The SAS statements needed to produce the following charts are found in the **Examples** section.

Frequency bar charts When you want to divide your data into groups based on the values of a variable, frequency bar charts are useful. For example, say you

have a data set containing information about employees. You want a bar chart comparing the number of male employees and the number of female employees. This chart is a frequency chart since it shows the number of employees (observations) of each sex. The following SAS code produces a vertical frequency chart:

```
PROC CHART;
   VBAR SEX;
```

Output 27.1 Vertical Frequency Bar Chart: The CHART Procedure



At the bottom of the chart are the two values of SEX, F and M. The vertical axis represents the number of observations in the data set containing the value; seventy-five of the employees are females and ninety-seven are males.

If you prefer a horizontal bar chart, use the HBAR statement instead of the VBAR statement. You can use the keyword HBAR in any of the following examples instead of VBAR. The following SAS code produces a horizontal frequency bar chart:

```
PROC CHART;
   HBAR SEX;
```

Output 27.2 Horizontal Frequency Bar Chart: The CHART Procedure

```
                                  EMPLOYEE SEX AND EDUCATION LEVEL                                    2

                                      FREQUENCY BAR CHART

SEX                                                                                  FREQ  CUM.   PERCENT   CUM.
                                                                                           FREQ             PERCENT
        |
F       |*************************************************************************      75    75    43.60    43.60
        |
M       |*****************************************************************************************  97   172    56.40   100.00
        |
        -----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+--
             5   10   15   20   25   30   35   40   45   50   55   60   65   70   75   80   85   90   95

                                          FREQUENCY
```

Percentage bar charts These charts are handy for showing what percentage of the observations falls into different groups. For example, say you want to show the percentage of employees of each sex. The TYPE=PERCENT option produces the percentage bar chart. The following SAS code produces a precentage bar chart:

```
PROC CHART;
   VBAR SEX / TYPE=PERCENT;
```

Output 27.3 Percentage Bar Chart: The CHART Procedure

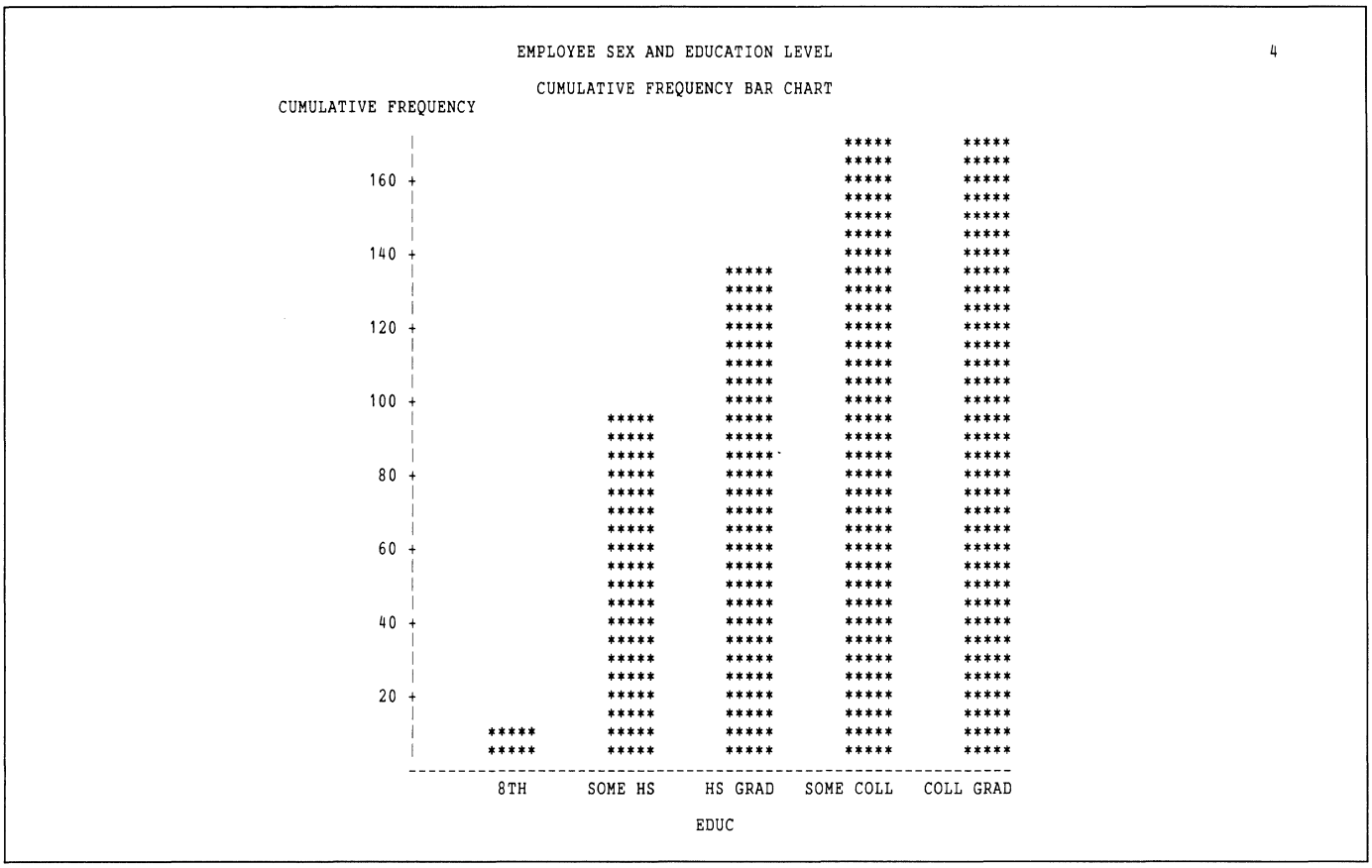
```
                                  EMPLOYEE SEX AND EDUCATION LEVEL                                    3

                                      PERCENTAGE BAR CHART
PERCENTAGE

                                    |                    *****
                                    |                    *****
                                    |                    *****
                                 50 +                    *****
                                    |                    *****
                                    |                    *****
                                    |          *****     *****
                                    |          *****     *****
                                 40 +          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                 30 +          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                 20 +          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                 10 +          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    |          *****     *****
                                    ------------------------
                                               F          M

                                             SEX
```

Cumulative frequency charts Sometimes you want a cumulative frequency chart where each bar represents the frequency of a given value plus the frequencies of all the values to its left in the chart. For example, using the data set contain-

ing employee information, you may want to show how many employees have not attended college. The TYPE=CFREQ option produces the cumulative frequency chart. The following SAS code produces a cumulative frequency bar chart:

```
PROC CHART;
   VBAR EDUC / TYPE=CFREQ;
```

Output 27.4 Cumulative Frequency Bar Chart: The CHART Procedure



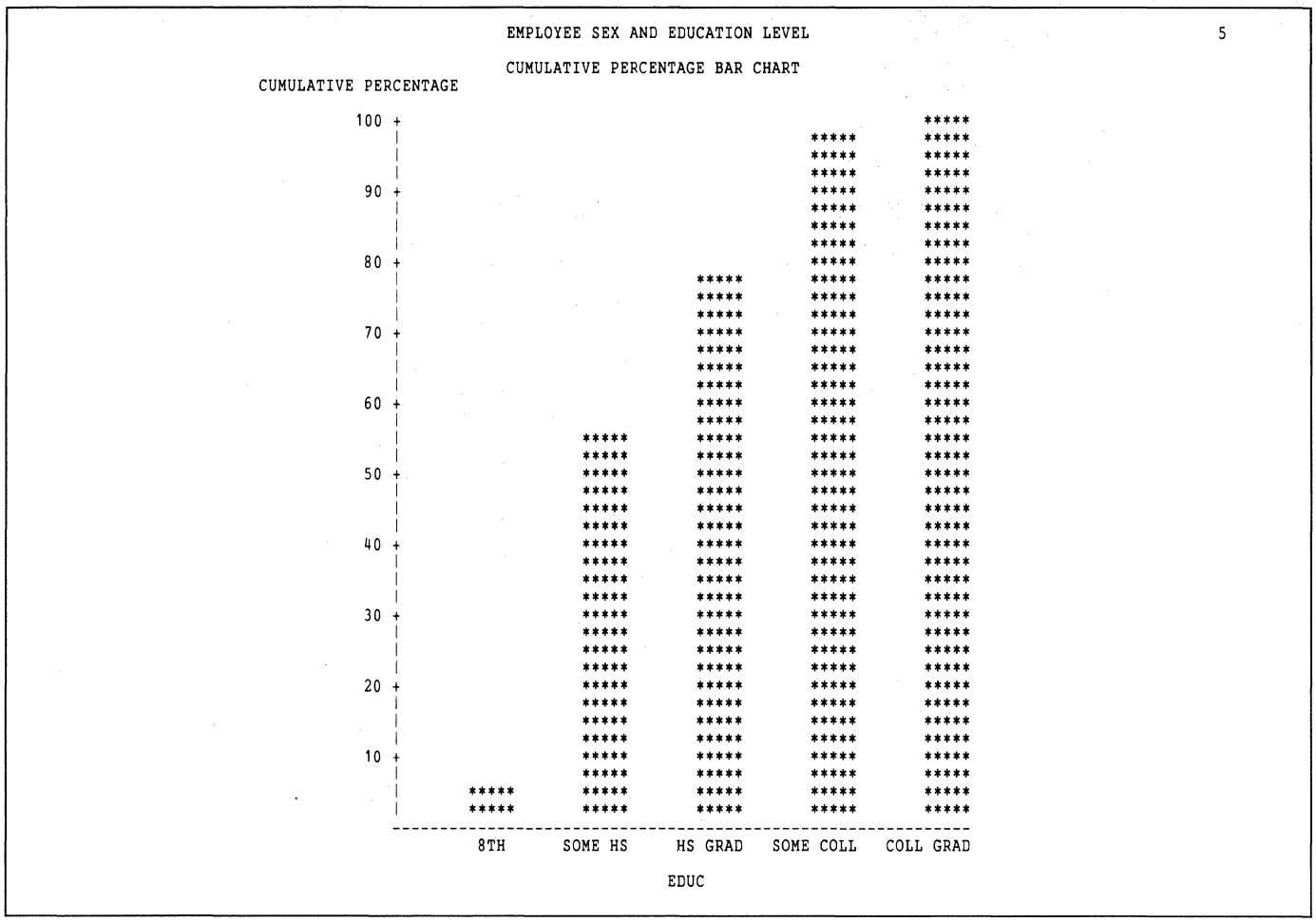
Output 27.4 shows that 135 employees have not attended college.

Cumulative percentage charts The bars of a cumulative percentage chart represent the percentage of the observations having a given value plus the percentages of all the values appearing to the left in the chart.

For example, suppose you want to represent the educational attainments of the employees in terms of percentages rather than frequencies. The TYPE=CPERCENT option produces the cumulative percentage chart. The following SAS code produces a cumulative percentage bar chart:

```
PROC CHART;
   VBAR EDUC / TYPE=CPERCENT;
```

Output 27.5 Cumulative Percentage Bar Chart: The CHART Procedure

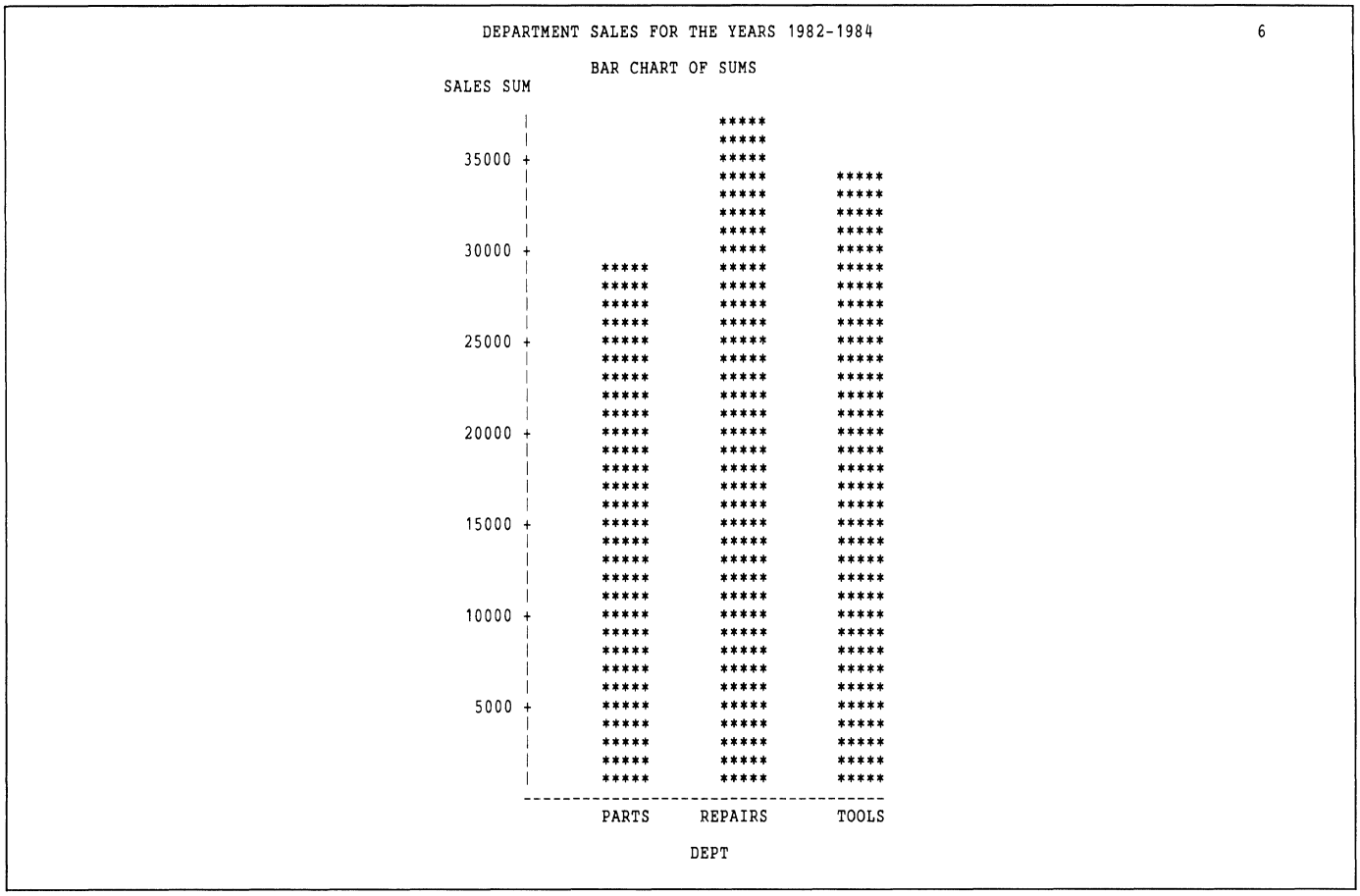


Bar charts of group totals These charts are like the frequency and percentage charts described earlier, but the vertical axis represents the group totals for another variable in the data set instead of frequencies or percentages. This variable is usually a continuous variable, such as SALES.

Suppose you have a data set that consists of twelve observations for each department. Each observation contains DEPT, the chart variable, which identifies the department, and the variable SALES, which gives each department's sales for that month. Suppose you want to see the total sales for the year for each department. You use the option SUMVAR= to specify SALES as the sum variable since you want the sum of each department's monthly sales for the chart. The following SAS code produces a bar chart of group totals:

```
PROC CHART;
   VBAR DEPT / SUMVAR=SALES;
```

Output 27.6 Bar Chart of Group Totals: The CHART Procedure



Suppose you have one observation per group—you already have each department's total sales and want to display them. For example, say your data set consists of three observations, each giving the total sales for one of the three departments. You can use exactly the same statements as above:

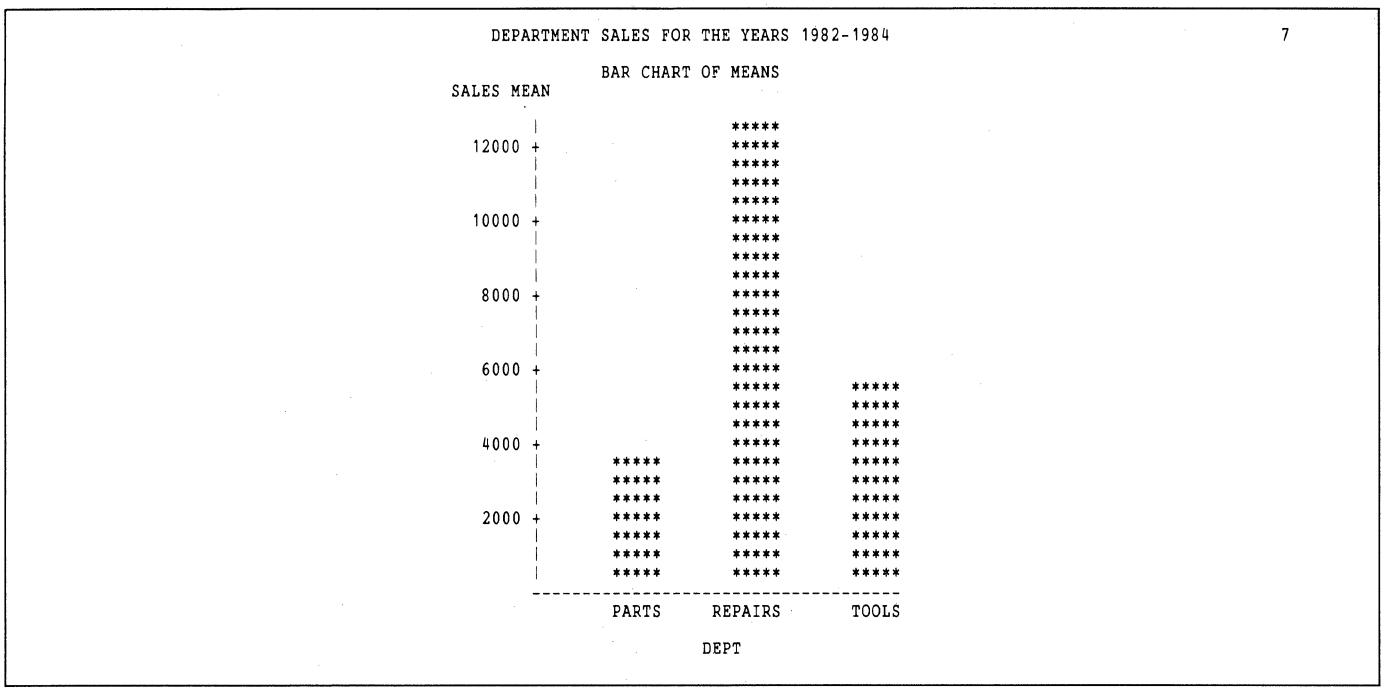
```
PROC CHART;
   VBAR DEPT / SUMVAR=SALES;
```

Even though PROC CHART does not need to sum the SALES values, they do represent a sum, and, thus, SUMVAR=SALES is the correct option to use. The results are like the chart above, but marked as VALUE rather than SUM. You can use this technique to present any value that has been entered as the value of a variable, such as a mean, standard deviation, or percentage.

Bar charts for group means Bar charts of group means are like the bar chart of group totals described earlier. However, the vertical axis represents the means of another variable rather than the sums. For example, to show each department's average monthly sales, use these statements to produce a bar chart for group means:

```
PROC CHART;
  VBAR DEPT / TYPE=MEAN SUMVAR=SALES;
```

Output 27.7 Bar Charts for Group Means: The CHART Procedure



Subdividing the bars You can show the distribution of a second variable by using the first character of each of the values to print the bars. For example, say you want to compare the number of males and females in each of the three departments. You can ask PROC CHART to subdivide the bars using F and M with these statements:

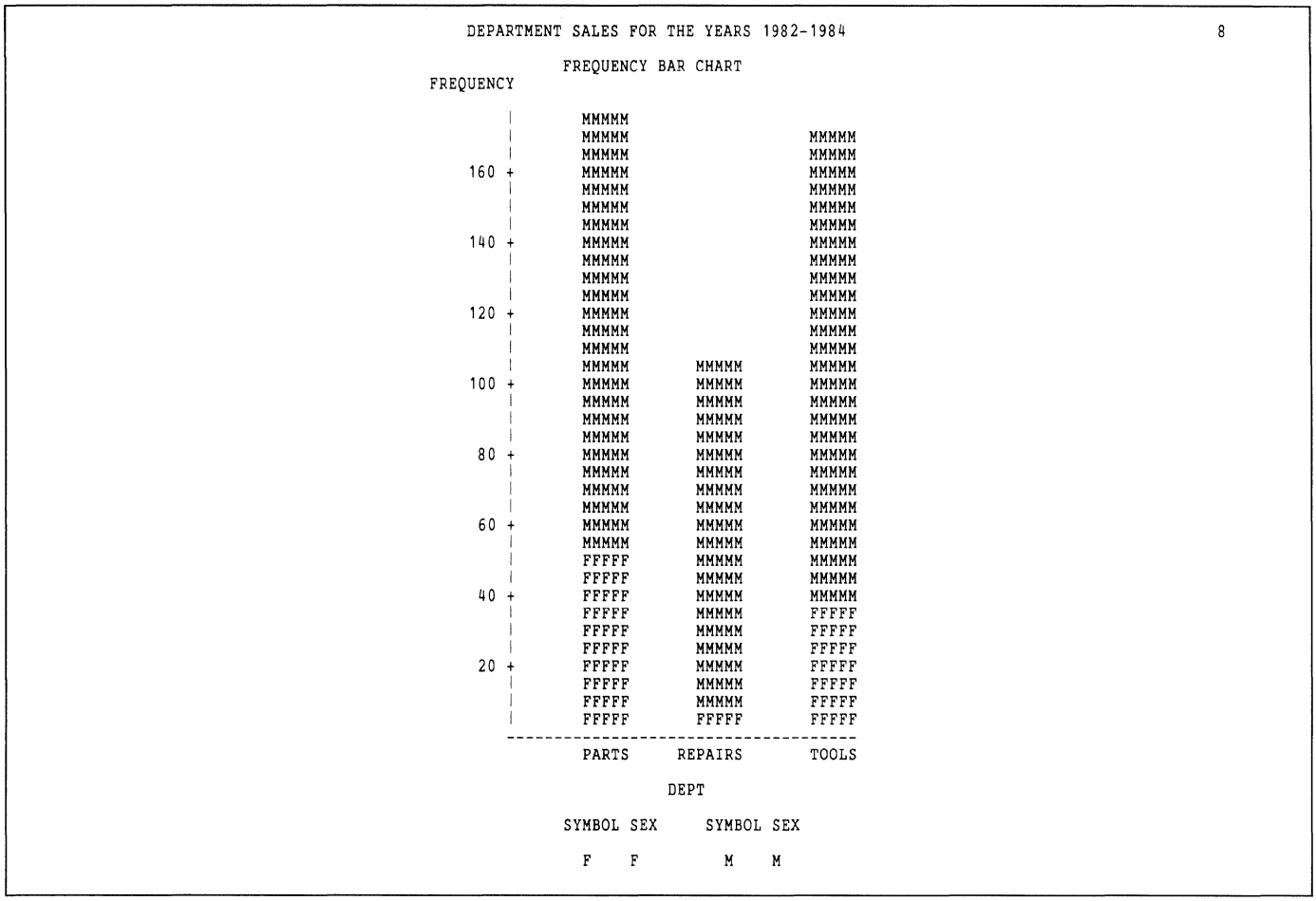
```
PROC CHART;
   VBAR DEPT / SUBGROUP=SEX;
```

From **Output 27.8**, you can see that females represent about one-third of the PARTS employees, none of the REPAIRS employees, and about one-fifth of the TOOLS employees.

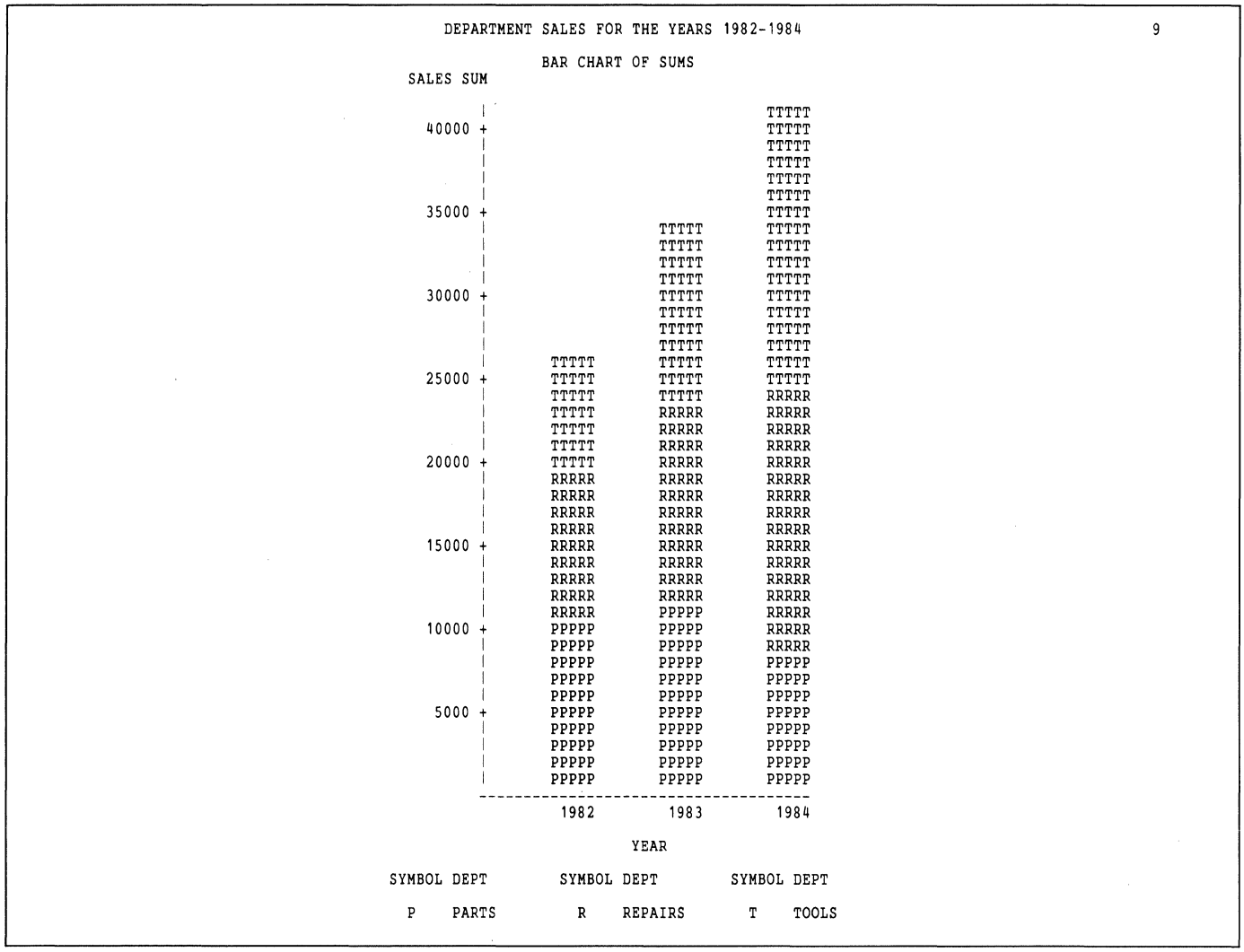
You can use other options in addition to SUBGROUP= to produce more complicated charts. For example, say you want to compare the sales figures for each of the three departments over the past three years. Your data set contains nine observations, each having values of the variables YEAR, DEPT, and SALES. These statements produce the chart shown in **Output 27.9**:

```
PROC CHART;
   VBAR YEAR / SUBGROUP=DEPT SUMVAR=SALES DISCRETE;
```

Output 27.8 Subdividing a Frequency Bar Chart: The CHART Procedure



Output 27.9 Subdividing a Bar Chart of Sums: The CHART Procedure

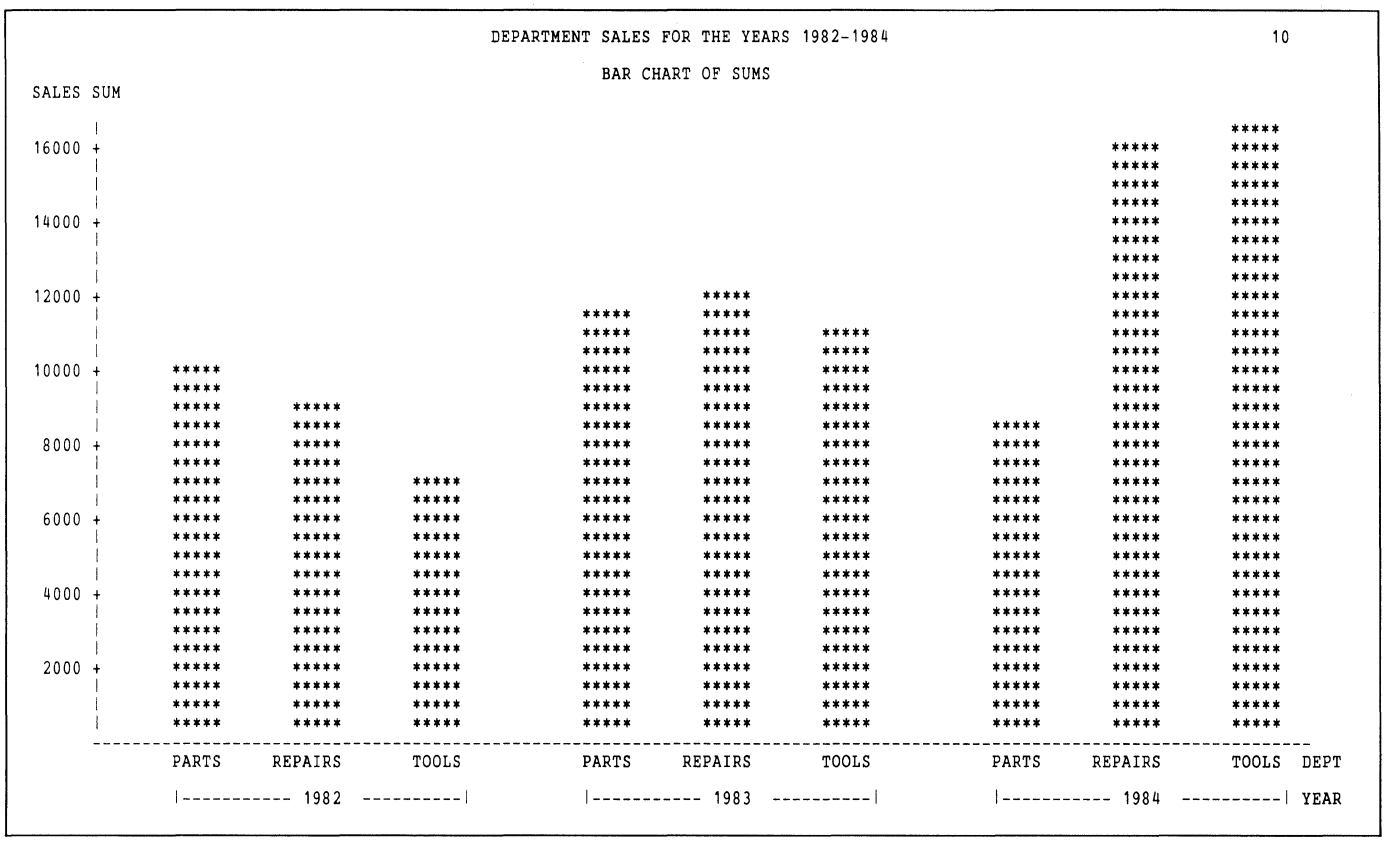


You need the DISCRETE option to display three distinct YEAR values. When DISCRETE is omitted for a numeric variable, the procedure divides the values into intervals. If you specify YEAR as a character variable, DISCRETE can be omitted.

Side-by-side charts Another way of comparing quantities is with side-by-side charts. You can show the information—total sales for the three departments over the past three years—with these statements:

```
PROC CHART;
   VBAR DEPT / SUMVAR=SALES GROUP=YEAR;
```

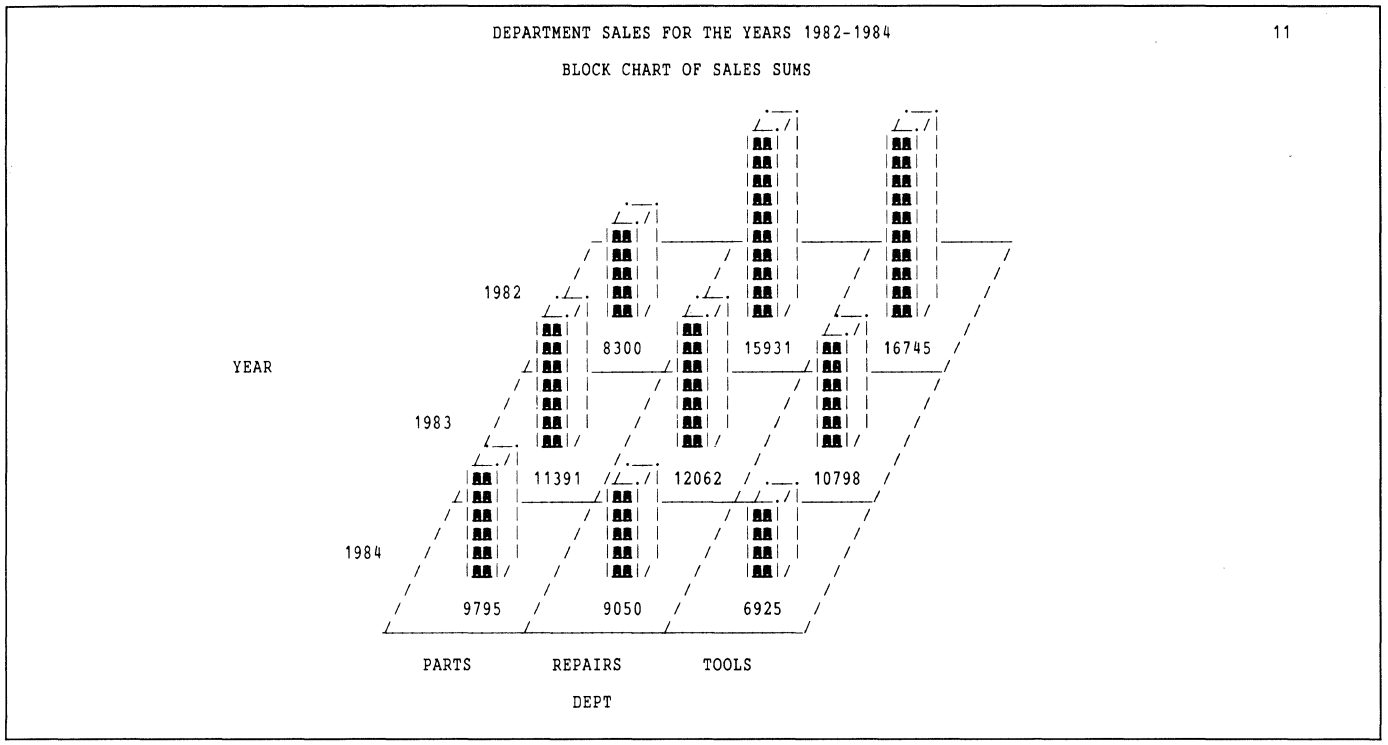
Output 27.10 Side-by-Side Charts: The CHART Procedure



Block charts Another way you can show the sales information by department for each year is with a block chart. Block charts look like city blocks with a building drawn in each block. You can represent the frequency, mean, or sum by the height of the building. Use the following SAS code to produce a block chart that displays the same total sales as the VBAR chart presented earlier:

```
PROC CHART;
   BLOCK DEPT / SUMVAR=SALES GROUP=YEAR DISCRETE;
```

Output 27.11 Block Chart of Sales Sums: The CHART Procedure



Pie charts You can also use the CHART procedure to draw a pie chart representing the distribution of a variable's values. For the example above where you want to show the total sales for each department over the three-year period, use these statements to produce a pie chart:

```
PROC CHART;
   PIE DEPT / SUMVAR=SALES;
```

Output 27.12 Sum Pie Chart of Sales Grouped by Department: The CHART Procedure

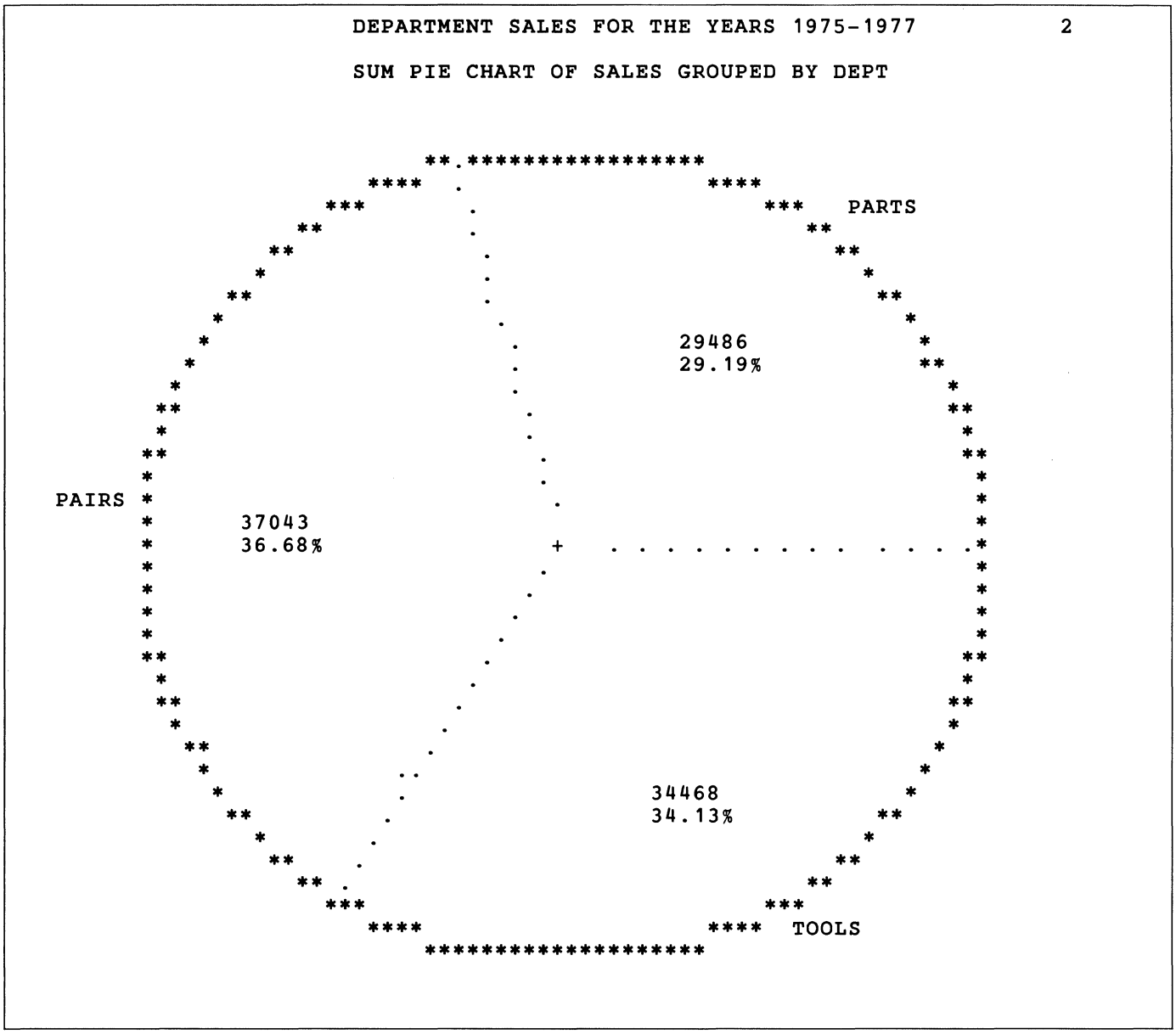
DEPARTMENT SALES FOR THE YEARS 1975-1977 1

OBS	DEPT	SEX	WT	P75	P76	P77	YEAR	SALES
1	PARTS	F	17	3500	2500	800	1975	3500
2	PARTS	F	17	3500	2500	800	1976	2500
3	PARTS	F	17	3500	2500	800	1977	800
4	PARTS	M	21	3651	5391	4500	1975	3651
5	PARTS	M	21	3651	5391	4500	1976	5391
6	PARTS	M	21	3651	5391	4500	1977	4500
7	PARTS	M	21	2644	3500	3000	1975	2644
8	PARTS	M	21	2644	3500	3000	1976	3500
9	PARTS	M	21	2644	3500	3000	1977	3000
10	TOOLS	F	12	5672	6100	7400	1975	5672
11	TOOLS	F	12	5672	6100	7400	1976	6100
12	TOOLS	F	12	5672	6100	7400	1977	7400
13	TOOLS	M	45	1253	4698	9345	1975	1253
14	TOOLS	M	45	1253	4698	9345	1976	4698
15	TOOLS	M	45	1253	4698	9345	1977	9345
16	REPAIRS	F	2	9050	12062	15931	1975	9050
17	REPAIRS	F	2	9050	12062	15931	1976	12062
18	REPAIRS	F	2	9050	12062	15931	1977	15931
19	REPAIRS	M	34	.	.	.	1975	.

(continued on next page)

(continued from previous page)

```
 20    REPAIRS     M     34        .         .         .     1976        .
 21    REPAIRS     M     34        .         .         .     1977        .
```

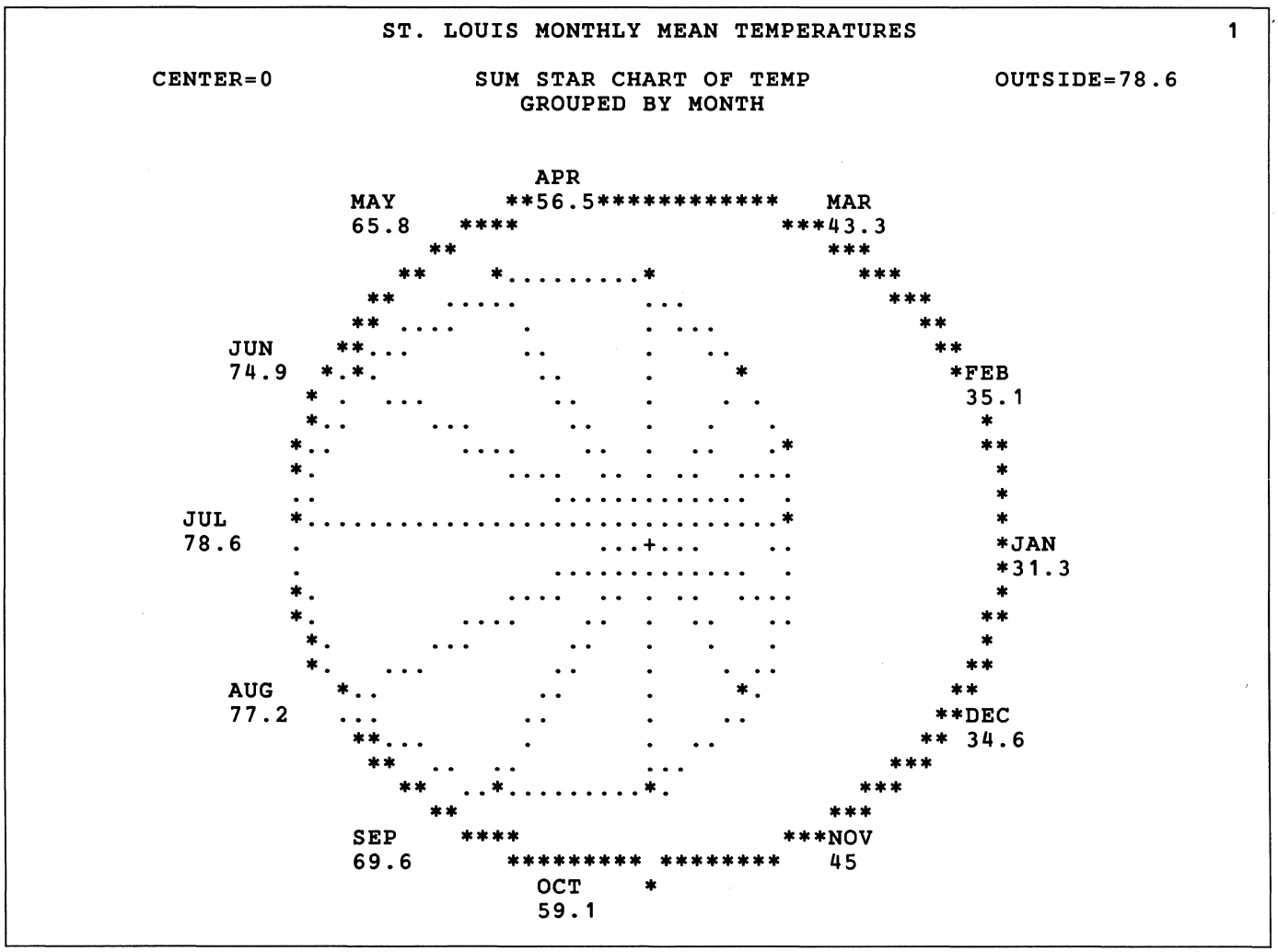


Star charts You can produce star charts showing group frequencies, totals, or means with PROC CHART. Star charts are appropriate for cyclical data, such as months of the year or hours of the day. The following SAS statements produce a star chart that shows monthly temperature averages. (The star chart that results is like a vertical bar chart where the bars are wrapped around in a circle and connected.) The statements are

```
PROC FORMAT;
   VALUE _MON 1=JAN  2=FEB  3=MAR  4=APR  5=MAY  6=JUN
              7=JUL  8=AUG  9=SEP 10=OCT 11=NOV 12=DEC;
```

```
DATA MONTHLY;
   TITLE 'ST. LOUIS MONTHLY MEAN TEMPERATURES';
   DO MONTH=1 TO 12; INPUT TEMP @@; OUTPUT; END;
   FORMAT MONTH _MON.;
   CARDS;
31.3 35.1 43.3 56.5 65.8 74.9 78.6 77.2 69.6 59.1 45.0 34.6
;

PROC CHART;
   STAR MONTH / SUMVAR=TEMP DISCRETE;
```

Output 27.13 Star Chart of Temperature Grouped by Month: The CHART Procedure



SPECIFICATIONS

The CHART procedure is controlled by the following statements:

PROC CHART *options;*
BY *variables;*
VBAR *variables / options;*
HBAR *variables / options;*
BLOCK *variables / options;*
PIE *variables / options;*
STAR *variables / options;*

Any number of chart request statements can follow a PROC CHART statement.

PROC CHART Statement

PROC CHART *options;*

The following options can be used in the PROC CHART statement:

DATA=*SASdataset* names the SAS data set to be used by PROC CHART. If DATA= is not specified, CHART uses the most recently created SAS data set.

LPI=*p* specifies the proportions of PIE and STAR charts. The value of LPI is

(lines per inch/columns per inch) * 10

and the default is LPI=6. For example, if you have a printer with 8 lines per inch and 12 columns per inch, specify LPI=6.6667. Also see **Printed Output** for pie and star charts.

BY Statement

BY *variables;*

A BY statement can be used with PROC CHART to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

VBAR Statement

VBAR *variables / options;*

In the VBAR statement, list the variables for which you want vertical bar charts. (The options with VBAR are described later.) If you list X as one of the chart variables, then a vertical bar chart is produced with the values of X underneath the bars.

Each chart takes one page. Along the vertical axis, PROC CHART describes the chart frequency, cumulative frequency, percentage, cumulative percentage, sum, or mean. At the bottom of each bar, CHART prints a value. 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, the options and parameters described below allow you to choose bar intervals and the number of bars, include missing values in the chart, produce side-by-side charts, and subdivide the bars.

The number of lines in the body of the VBAR chart is limited to sixty-six, regardless of the system PAGESIZE option in effect. If the number of characters per line (LINESIZE) does not allow at least one blank space between bars, an error message is printed and PROC CHART produces an HBAR chart instead of a VBAR chart, unless the NOSPACE option is included.

HBAR Statement

HBAR *variables / options;*

The HBAR statement requests a horizontal bar chart for each variable listed. For example, the statements

```
PROC CHART;
   HBAR A X1 X2;
```

produce three horizontal bar charts. Each chart occupies one or more pages, depending on the number of levels; each bar is one line. (The options available in the HBAR statement are discussed later.)

BLOCK Statement

BLOCK *variables / options;*

In the BLOCK statement, list the variables for which you want block charts. Since each block chart must fit on one output page, there are some restrictions on the number of levels of the BLOCK and GROUP variables. (BLOCK statement options are described later.)

Table 27.1 shows the maximum number of levels of GROUP and BLOCK variables for selected LINESIZEs that can be represented in a block chart using a sixty-six-line page.

Table 27.1 Maximum Number of Levels of GROUP and BLOCK Variables: The CHART Procedure

groups	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, the maximum number of BLOCK levels that can be produced may be reduced by one. BLOCK level values are truncated to twelve characters. If these limits are exceeded, data are represented as an HBAR chart.

PIE Statement

PIE *variables / options;*

The PIE statement requests a pie chart for each variable listed. For example, the statements

```
PROC CHART;
   PIE A X1 X2;
```

produce three one-page pie charts. (Options available for PIE charts are discussed later.)

PROC CHART determines the number of slices for the pie in the same way that it determines the number of bars for VBAR charts. Any slices of the pie accounting for less than three print positions are lumped together into a slice called OTHER.

The pie's size is determined only by the LINESIZE and PAGESIZE system parameters. The pie looks elliptical if your printer does not print 6 lines per inch and 10 columns per inch. If your printer prints 8 lines per inch and 10 columns per inch, specify LPI=8 in the PROC CHART statement:

```
PROC CHART LPI=8;
```

If a PIE chart is requested for a variable with over fifty levels, an HBAR chart is produced instead.

STAR Statement

STAR variables / options;

The STAR statement requests a star chart for each variable listed. (Options on the STAR statement are discussed later.) For example, the statements

```
PROC CHART;
   STAR Z;
```

produce a one-page star chart for the variable Z.

The number of points in the star is determined in the same way as the number of bars for VBAR charts.

If all the data to be charted with a STAR statement are positive, the center of the star represents zero and the outside circle represents the maximum value. If negative values occur in the data, the center represents the minimum. See the AXIS= option below for more information about how to specify maximum and minimum values. If a STAR chart is requested for a variable with over twenty-four levels, an HBAR chart is produced instead.

Note: if you want different variables to form the rays of the star, use an OUTPUT statement in a DATA step to create new observations having one variable with values equal to the variables you want represented by the rays; create another variable whose values are the original variable names.

Options for VBAR, HBAR, BLOCK, PIE, and STAR Statements

If any of the options below are used, a slash (/) precedes the option keywords. The slash is not necessary if no options are specified.

The following options can be used with all the types of charts in the CHART procedure: vertical bar, horizontal bar, block, pie, or star charts.

MISSING
: specifies that missing values are to be considered as valid levels for the chart variable.

DISCRETE
: is used when the numeric chart variable specified is discrete rather than continuous. If DISCRETE is omitted, PROC CHART assumes that all numeric variables are continuous. If the MIDPOINTS= or LEVELS= options are not specified, the procedure automatically chooses the intervals for the chart.

TYPE=FREQ
TYPE= PERCENT | PCT
TYPE=CFREQ
TYPE=PERCENT | CPCT
TYPE=SUM
TYPE=MEAN
: specifies what the bars or sections in the chart represent. If the TYPE= option is omitted, the default TYPE is FREQ. When SUMVAR= is specified, the default TYPE is SUM.

TYPE=FREQ
: makes each bar or section represent the frequency with which a value or range occurs for the chart variable in the data.

TYPE=PERCENT or PCT
: makes each bar or section represent the percentage of observations of the chart variable having a given value or falling into a given range.

TYPE=CFREQ
: makes each bar or section represent cumulative frequency, which is the frequency for the group plus all the frequencies shown to its left.

TYPE=CPERCENT or CPCT
: makes each bar represent the cumulative percentage of observations of the chart variable, which is the percentage of the group plus the percentages of the bars shown to its left.

TYPE=SUM
: makes each bar or section represent the sum of the SUMVAR= variable for observations having the bar's value. For example, the statement

```
VBAR DEPT / SUMVAR=SALES;
```

: produces a chart with one bar or section for each DEPT value. The bar height for a given DEPT corresponds to the total of the SALES values for observations having that DEPT value.

TYPE=MEAN
: makes each bar or section represent the mean of the SUMVAR= variable for observations having the bar's value.

SUMVAR=*variable*
: names the variable to collect summaries for means, sums, or frequencies. The SUMVAR= option is useful for producing bar charts showing total expenditures for each department, or means at each level of an experiment. For example, the statement

```
VBAR LOCATION / TYPE=MEAN SUMVAR=YIELD;
```

: produces a chart showing the mean yield for each location. Another example

```
VBAR DEPT / SUMVAR=EXPEND;
```

charts total expenditures by department. If SUMVAR is specified but TYPE= is not MEAN or SUM, then TYPE=SUM overrides whatever TYPE= value is specified.

MIDPOINTS=*values*
defines the range of values for the chart variable each bar or section represents by specifying the range midpoints. For example, the statement

```
VBAR X / MIDPOINTS=10 20 30 40 50;
```

produces a chart with five bars: the first bar represents the range of data values with a midpoint of ten; the second bar represents the range of data values with a midpoint of twenty; and so on. When the variables given in the VBAR statement are numeric, the midpoints must be given in ascending order, although they need not be uniformly distributed. For example, the statement

```
VBAR X / MIDPOINTS=10 100 1000 10000;
```

produces a chart of X with logarithmic intervals. Numeric MIDPOINTS lists of the form

```
MIDPOINTS=10 TO 100 BY 5
```

are also acceptable.
For character variables, MIDPOINTS can be specified in any order, which is useful in ordering the bars or in specifying a subset of the possible values. For example, you can give a list of the form

```
MIDPOINTS='JAN' 'FEB' 'MAR'
```

Without the MIDPOINTS= option, the values are displayed in sorted order.

FREQ=*variable*
is used when a variable in the data set represents a count (or weight) for each observation. Normally, each observation contributes a value of one to the frequency counts. When FREQ appears, each observation contributes the FREQ variable's value. If the FREQ variable's values are not integers, they are truncated to integers. If the values are missing or negative, the contribution is zero. If SUMVAR= is specified, the sums are multiplied by the FREQ value.

AXIS=*value*
specifies the maximum value to use in constructing the FREQ, PCT, CFREQ, CPCT, SUM, or MEAN axis. If the VBAR or HBAR chart is of TYPE=SUM or TYPE=MEAN and if any of the sums or means are less than zero, then a negative minimum value can also be specified in the AXIS= option. Otherwise, a minimum value of zero is always assumed. Counts or percentages outside the maximum (or minimum) override the AXIS= specification. If AXIS= is specified and a BY statement also appears, uniform axes are produced over BY groups. When AXIS= appears in a STAR statement, the first value specified is the center (minimum) of the star and the second value is the outside circle (maximum). If only one AXIS= value is specified in the STAR statement, PROC CHART assumes this value as the maximum and zero as the minimum. For example, the statements

```
PROC CHART;
   STAR A / SUMVAR=X TYPE=SUM AXIS=100 200;
```

produce a star chart for the sums of X classified by A and scaled from 100 at the center to 200 at the outside circle.

More Options for VBAR, HBAR, and BLOCK Statements

GROUP=*variable*

produces side-by-side charts, with each chart representing the observations having a given value of the GROUP= variable. The GROUP= variable can be character or numeric and is assumed to be discrete. For example, the statement

```
VBAR SEX / GROUP=DEPT;
```

produces a frequency bar chart for males and females in each department. Missing values for a GROUP variable are treated as valid levels when a chart is produced.

SUBGROUP=*variable*

requests that each bar be subdivided into characters that show the SUBGROUP= variable's contribution to the bar. For example, the statement

```
VBAR DEPT / SUBGROUP=SEX;
```

produces a chart with one bar for each department. The portion of each bar filled in with the character M represents those observations that have a SEX value of M.

The first character of the value is used to fill in the portion of the bar corresponding to the value, unless more than one value begins with the same first character. In that case, the letters A, B, C, and so on are used. The characters used in the chart and the values they represent are given in a table at the bottom of the chart.

Missing values for a SUBGROUP= variable are treated as valid levels when a chart is produced.

PROC CHART calculates the height of the bar for each subgroup individually and then rounds each bar's percentage of the total bar up or down. Thus, the total height of the bar may be higher or lower than the same bar without SUBGROUP=.

With both TYPE=MEAN and the SUBGROUP= option, PROC CHART first calculates the mean for each bar and then subdivides the bar into the percentages contributed by each subgroup.

LEVELS=*n*

specifies the number of bars representing each chart variable when the variables given in the VBAR statement are continuous.

SYMBOL='*char*'

defines the symbol to be used in the body of standard HBAR and VBAR charts with no subgrouping. The default SYMBOL value is the asterisk (*).

If overprinting is available on your printer, you can use SYMBOL= to specify two or three characters that will be printed over each other for the symbol. For example, specifying SYMBOL='XOA' means that the three characters X, O, and A are printed on top of each other to make up the symbol.

For IBM 3800 printers that can print shaded characters, you can specify the symbol as a hex constant. The hex characters are enclosed in quotes, followed by the letter X:

```
VBAR DEPT / SYMBOL='A5'X;
```

NOSYMBOL
suppresses the subgrouping symbol table printed below the chart for VBAR and HBAR charts.

NOZEROS
specifies that any bar with zero value be suppressed.

G100
is used in conjunction with the GROUP= option to force the bars and statistics to add to 100% for each group.

Options for VBAR and HBAR Statements

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

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

REF=*value*
requests that a single reference line be drawn on the response axis. For TYPE=FREQ or TYPE=CFREQ, this REF= value should be a frequency; for TYPE=PCT or TYPE=CPCT, the REF= value should be a percent between 1 and 100. For TYPE=SUM or TYPE=MEAN, the REF= value should be a sum or mean.

More HBAR Statement Options

These options are used only in the HBAR statement:

NOSTAT
specifies that no statistics be printed with a horizontal bar chart.

FREQ
specifies for a horizontal bar chart that the frequency of each bar be printed to the side of the chart.

CFREQ
specifies that the cumulative frequency be printed.

PERCENT
specifies that the percentages of observations having a given value for the chart variable be printed.

CPERCENT
specifies that the cumulative percentages be printed.

SUM
specifies that the total number of observations that each bar represents be printed.

MEAN
specifies that the mean of the observations represented by each bar be printed.

For charts produced with any TYPE= specification without a SUMVAR= variable, PROC CHART can print FREQ, CFREQ, PERCENT, and CPERCENT.

For TYPE=MEAN with a SUMVAR= variable, PROC CHART can print FREQ and MEAN. For a TYPE=SUM specification, PROC CHART can print FREQ and SUM.

Another VBAR Statement Option

This option is used only on the VBAR statement:

NOSPACE
: specifies that if the LINESIZE does not allow room for spaces between the bars, PROC CHART can print a VBAR chart without spaces between bars. If space is still insufficient, an HBAR chart is printed instead.

DETAILS

Missing Values

The way that PROC CHART treats missing values depends on whether or not the MISSING option is specified. See the section **Options for VBAR, HBAR, BLOCK, PIE, and STAR Statements** above.

EXAMPLES

Employee Sex and Education Level Data: Example 1

The statements below produced the data used in the CHART procedure for the charts on employee sex and education.

These statements include a DATA step to enter the values and the FORMAT procedure to define a format for the EDUC variable (for more information, see the chapter on PROC FORMAT):

```
PROC FORMAT;
   VALUE _E 1=8TH 2=SOME HS 3=HS GRAD 4=SOME COLL 5=COLL GRAD;
DATA WSEXED;
   TITLE 'EMPLOYEE SEX AND EDUCATION LEVEL';
   INPUT SEX $ EDUC @@;
   FORMAT EDUC _E.;
   CARDS;
M 1  M 2  M 2  M 1  M 4  M 5  M 3  F 2  F 2  M 1  F 4
more data lines
;
```

Department Sales for the Years 1982-1984: Example 2

These statements produced the data for the charts on departmental sales:

```
DATA SALES;
   TITLE 'DEPARTMENT SALES FOR THE YEARS 1982-1984';
   INPUT DEPT :$7. SEX :$1. WT P82 P83 P84;
   YEAR=1982; SALES=P82; OUTPUT;
   YEAR=1983; SALES=P83; OUTPUT;
   YEAR=1984; SALES=P84; OUTPUT;
   CARDS;
PARTS   F 17   3500  2500   800
PARTS   M 21   3651  5391  4500
PARTS   M 21   2644  3500  3000
TOOLS   F 12   5672  6100  7400
TOOLS   M 45   1253  4698  9345
REPAIRS F  2   9050 12062 15931
more data lines
;
```

Sales Statistics from PROC SUMMARY: Example 3

The statements below produce charts from summary statistics for three variables: PRODUCT with three levels, YEAR with three levels, and DATE. An example in "PROC Step Applications" shows the DATA step that reads these data into a SAS data set (named A) and the PROC SUMMARY step that produces data set B containing the summary statistics.

PROC CHART displays charts of a subset of the summary statistics contained in data set C.

The first chart is a horizontal bar chart showing SALES for each PRODUCT by DATE. You need the DISCRETE option to display each value of DATE rather than the midpoints of the DATE values. The next chart is a block chart showing number of products by year. The pie chart of PRODUCT uses SALES as a weighting variable specified with the FREQ= option. The final vertical bar chart by YEAR shows number of products as the SUBGROUP= variable. You must specify DISCRETE so that the chart does not show YEAR with continuous values.

```
DATA C;
   SET B;
   IF _TYPE_=3;
PROC CHART;
   HBAR DATE / SUMVAR=SALES GROUP=PRODUCT DISCRETE;
   BLOCK PRODUCT / GROUP=YEAR;
   PIE PRODUCT / FREQ=SALES;
   VBAR YEAR / DISCRETE SUBGROUP=PRODUCT;
```

Chapter 28

The COMPARE Procedure

Operating systems: All

ABSTRACT

PROC COMPARE compares the values of variables in two SAS data sets and reports the differences found. PROC COMPARE is especially useful in data base management applications (to verify data updates by comparing "before" and "after" images of a data base) and in forecasting (to compare forecasts made under different assumptions).

INTRODUCTION

The COMPARE procedure matches variables and observations in a base data set with those in a comparison data set. For numeric variables, PROC COMPARE computes the difference and percent difference of each pair of values, and if the difference is large enough, judges the pair to be unequal. PROC COMPARE then prints a list of variable pairs that were compared and found equal for all observations and a separate list of those with at least one unequal observation and the number of observations unequal for each. For each unequal variable, a report is printed showing the unequal values found. For numeric variables, the difference, percent difference, and summary statistics for each are shown.

Normally, the corresponding variables in the base and comparison data sets have the same names, but COMPARE can also compare variables with differing names. Different variables within one data set can also be compared.

COMPARE can also produce an output data set containing the differences or percent differences of the numeric variables compared.

SPECIFICATIONS

PROC COMPARE *options*;
VAR *variables*;
WITH *variables*;
ID *variables*;
BY *variables*;

PROC COMPARE Statement

PROC COMPARE *options*;

The following options work on all operating systems:

DATA=*SASdataset*
: names the data set to be used as the base data set for the comparison. If DATA= is omitted, the most recently created SAS data set is used.

COMPARE=*SASdataset*
: names the data set to be used as the comparison data set for the comparison. If COMPARE= is omitted, the DATA= data set is used and the comparison data set is the same as the base data set.

OUT=*SASdataset*
: requests that the differences for the numeric variables be written to a SAS data set. If OUT= is omitted, no output data set is created.

OUTPERCENT
: requests that the percent differences, instead of the differences, be written to the OUT= data set. OUTPERCENT is ignored unless OUT= is used.

OUTNOEQUAL
: requests that the differences or percent differences written to the OUT= data set be set to missing values for values that compared equal. OUTNOEQUAL is ignored unless OUT= is used.

CRITERION=*n*
: specifies the criterion for judging the equality of numeric values. See the section on **The Equality Criterion** in this chapter for details. The default value is CRITERION=.001.

METHOD=*name*
: specifies the method for judging the equality of numeric values. The methods available are RELATIVE, PERCENT, and ABSOLUTE. See the section on **The Equality Criterion** for details. The default is METHOD=RELATIVE.

ALLOBS
: requests that values and differences for all matching observations be printed. If ALLOBS is omitted, values are printed only for observations judged unequal.

NOOBS
: suppresses the listing of the value pairs and their differences.

ALLVARS
: requests that a detailed report of the value pairs be printed for each variable. (ALLVARS implies the ALLOBS option.) If ALLVARS is omitted, reports are generated only for variable pairs that compare unequal.

STATS
: requests the printing of summary statistics for the numeric variable pairs that compare unequal.

ALLSTATS
: requests the printing of summary statistics for all numeric variable pairs.

NOSUMMARY
: suppresses the printing of the summary notes, the list of variables that compare equal, and the list of variables that compare unequal.

NOLISTEQUAL
: suppresses the printing of the list of variables that compare equal.

NOPRINT
: suppresses all printed output. (Used in conjuction with the OUT= option.)

FUZZ=*n*
: requests that values of smaller magnitude than *n* not be printed on the reports. The default is FUZZ=1E-12.

NOMISSING or NOMISS
: specifies that missing values are to compare equal to any value. If NOMISSING is omitted, a missing value compares equal only to another missing value of the same kind.

NOMISS1
: specifies that the NOMISSING option is to apply for values in the base data set but not for values in the comparison data set.

NOMISS2
: specifies that the NOMISSING option is to apply for values in the comparison data set but not for values in the base data set.

VAR Statement

VAR *variables;*

The VAR statement lists the variables in the base data set to be compared with matching variables in the comparison data set. If no VAR statement is used, all variables in the base data set are compared except those appearing on the BY and ID statements. If no matching variable in the comparison data set is found for a variable in the VAR list, a warning is printed on the SAS Log and the variable is ignored.

WITH Statement

WITH *variables;*

If the variables to be compared have different names in the comparison data set from those in the base data set, specify these names using the WITH statement. The first variable in the WITH statement list corresponds to the first variable in the VAR statement list, the second with the second, and so on. If the WITH list is shorter than the VAR list, the extra variables in the VAR list are assumed to have

the same names in the comparison data set. If the WITH list is longer than the VAR list, the extra variables in the WITH list are ignored.

A VAR statement should be used when a WITH statement is used. If the WITH statement is omitted, the variables to be compared are assumed to have the same names in the comparison data set as in the base data set. If the COMPARE= option is omitted from the PROC COMPARE statement, a WITH statement **must** be used.

A variable name can appear any number of times on the VAR statement and on the WITH statement, and by selecting VAR and WITH lists, the variables can be compared in any permutation.

ID Statement

ID *variables . . .*;

The ID statement lists variables that are used to match observations in the base data set with corresponding observations in the comparison data set. The ID statement also lists variables that are used to identify observations on the reports listing variable values and differences. If an ID variable is not found in the comparison data set, a warning is printed on the SAS Log and the variable is not used to match observations in the comparison data set. (It is used to identify observations on the reports and is output to the OUT= data set.) All of the ID variables must be in the base data set, and both data sets must be sorted by the ID variables in common (within the BY variables, if any).

If the ID statement is omitted, the first observation in the base data set is matched with the first observation in the comparison data set, the second with the second, and so on.

BY Statement

BY *variables. . .*;

A BY statement can be used with PROC COMPARE. Both the base and comparison data sets must be sorted by the BY variables.

DETAILS

Output Data Set (OUT=)

The output data set contains the BY and ID variables and the numeric VAR statement variables for which matching variables were found in the comparison data set. Variable names and attributes in the output data set are taken from the base data set. The values of the VAR statement variables in the output data set are the differences (the comparison value less the base value), or the percent differences if the OUTPERCENT option is specified.

If the OUTNOEQUAL option is used, the special missing value *.E* will be output in place of differences that were judged "equal" (see below). The output data set contains an observation for each observation in the base data set for which a matching observation was found in the comparison data set. Nonmatching observations are not output.

The Equality Criterion

PROC COMPARE judges numeric values to be "equal" if their difference is sufficiently small relative to the CRITERION value. PROC COMPARE provides three methods for applying the CRITERION:

- The RELATIVE method uses the absolute relative difference.
- The PERCENT method uses the absolute percent difference.
- The ABSOLUTE method uses the absolute difference.

For a numeric variable compared, let *x* be its value in the base data set (DATA=), and let *y* be its value in the comparison data set (COMPARE=). If both *x* and *y* are nonmissing, the values are judged unequal according to the METHOD and CRITERION as described below.

```
METHOD=RELATIVE or Default:
    unequal if ABS(y-x)/((ABS(x)+ABS(y))/2) > CRITERION
    equal   if x=y=0

METHOD=PERCENT:
    unequal if ABS(y-x)/ABS(x)*100          > CRITERION, for x¬=0
         or if y¬=0                                      for x=0

METHOD=ABSOLUTE:
    unequal if ABS(y-x)                     > CRITERION
```

If *x* or *y* is missing, then the comparison depends on the NOMISSING option. If the NOMISSING option is in effect, a missing value will always compare 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 CRITERION specified is negative, then the actual criterion used is made equal to the absolute value of the CRITERION option times a very small number ε 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. This means that CRITERION=-1000 will often provide a reasonable test of the equality of computed results at the machine level of precision.

For character variables, if one value has a greater length than the other, 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, blank character values compare equal to anything.

Difference and Percent Difference

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 the difference and percent difference are defined as:

$$\text{Difference} = y - x$$

$$\text{Percent Difference} = (y - x)/x * 100, \text{ for } x\neg = 0$$

$$\text{or} = \text{missing, for } x = 0$$

Memory Limitations

For each BY group, PROC COMPARE reads the values for the base variables into memory and then reads and matches the comparison values. Thus, when comparing very large data sets, memory limitation problems may occur. If you receive error messages for insufficient memory, increase the memory available to SAS or move the first few ID variables to a BY statement to force PROC COMPARE to process the data in BY groups.

Printed Output

PROC COMPARE can produce three kinds of reports:

1. A report containing summary notes for the comparisons and a listing of the variables that compare equal and of those that compare unequal and the number of unequal observations (NDIF) found for each. This report is suppressed by the NOSUMMARY option.
2. Reports (printed separately for each variable pair) of the differences found at each observation. The report includes: the ID variable values for the observation, the value in the base data set, the value in the comparison data set, and, for numeric variables, the difference and percent difference. For character variables, only the first 20 characters of the values are printed. Unless the ALLOBS option is used, only the observations for which the values were judged unequal will be listed.
3. For numeric variables, a table of summary statistics is printed when the STATS or ALLSTATS option is used. First, the number of observations unequal (NDIF) and the percent of the matching observations unequal are printed. Second, the correlation of the base and comparison values (R) and the square of the correlation (RSQ) are printed. Next, descriptive statistics are printed for the base values, comparison values, differences, and percent differences.

   The statistics printed for each are N (number nonmissing), MEAN, STD, MAX, MIN, STDERR of MEAN, T ratio, and probability of greater |T| if true MEAN=0. These statistics are computed from all of the nonmissing matching observations for the item, but the R and RSQ statistics are computed from the matching observations that are nonmissing for **both** the base and comparison values. When the STATS option is used, summary statistics will be reported only for the variables judged unequal.

   The summary statistics reports are combined with the listing of differences (described above) when both are produced. (Note that the statistics are calculated from all the nonmissing matching observations and not just from the unequal observations printed in the listing of differences.)

EXAMPLES

Finding the Differences between Two Data Sets: Example 1

Given monthly data sets ONE and TWO, find the differences between them for variables A, B, and C.

```
PROC PRINT;
   TITLE 'DATA SET ONE';
PROC PRINT;
   TITLE 'DATA SET TWO';
PROC COMPARE DATA=ONE COMPARE=TWO;
   VAR A B C;
   ID YEAR MONTH;
   TITLE;
```

Output 28.1 Finding the Differences between Two Data Sets

DATA SET ONE 1

OBS	YEAR	MONTH	A	B	C
1	1984	1	1	2	JAN.
2	1984	2	4	5	FEB.
3	1984	3	7	8	MARCH
4	1984	4	10	11	APR.
5	1984	5	13	14	MAY
6	1984	6	16	17	JUNE
7	1984	7	19	20	JULY
8	1984	8	22	23	AUGUST

DATA SET TWO 2

OBS	YEAR	MONTH	A	B	C
1	1984	1	1	2.0000	JAN.
2	1984	2	4	5.0000	FEB.
3	1984	3	7	8.0989	MARCH
4	1984	4	10	10.0157	+FOUR
5	1984	5	13	15.4770	MAY
6	1984	6	16	16.8609	XXXXX
7	1984	7	19	20.4357	JULY
8	1984	8	22	22.3798	AUGUST

3

COMPARE PROCEDURE
COMPARISON OF ONE WITH TWO

COMPARISON SUMMARY

DATASET	OBS	VARIABLES	COMPARISON RESULTS	
ONE	8	5	EQUAL	1
TWO	8	5	UNEQUAL	2
MATCHED	8	5	TOTAL	3

COMPARISONS EQUAL

VARIABLE	TYPE	LEN	LABEL
A	NUM	8	A NUMBERIC VARIABLE "A"

COMPARISONS UNEQUAL

VARIABLE	TYPE	LEN	LABEL	NDIF
B	NUM	8		6
C	CHAR	8	A CHARACTER VARIABLE "C"	2

```
                          COMPARE PROCEDURE                                              4
                      COMPARISON OF ONE WITH TWO

 ----------------------------------------------------------------------
                        ||  VARIABLE     COMPARE
     YEAR       MONTH   ||         B           B       DIFF.     % DIFF
  ________   ________   ||  ________    ________    ________   ________
                        ||
   1984.00    3.00000   ||   8.00000     8.09886     0.09886    1.23581
   1984.00    4.00000   ||  11.00000    10.01573    -0.98427   -8.94789
   1984.00    5.00000   ||  14.00000    15.47700     1.47700   10.55001
   1984.00    6.00000   ||  17.00000    16.86094    -0.13906   -0.81798
   1984.00    7.00000   ||  20.00000    20.43574     0.43574    2.17872
   1984.00    8.00000   ||  23.00000    22.37975    -0.62025   -2.69673
 ----------------------------------------------------------------------

 ----------------------------------------------------------------------
                        || A CHARACTER VARIABLE "C"
                        || VARIABLE                COMPARE
     YEAR       MONTH   || C                       C
  ________   ________   || ________                ________
                        ||
   1984.00    4.00000   || APR.                    +FOUR
   1984.00    6.00000   || JUNE                    XXXXX
 ----------------------------------------------------------------------
```

Verifying Changes Made to a Master File: Example 2

Given a quarterly master file and a data set with update transactions, verify the changes made to the master file by the update.

```
PROC COMPARE DATA=NEW COMPARE=DATABASE.MASTER NOMISS1;
   ID YEAR QTR;
   TITLE 'COMPARISON OF UPDATE DATA WITH DATA BASE BEFORE UPDATE';

DATA DATABASE.MASTER;
   UPDATE DATABASE.MASTER NEW;
   BY YEAR QTR;
```

Output 28.2 Verifying Changes Made to a Master File

```
              COMPARISON OF UPDATE DATA WITH DATA BASE BEFORE UPDATE                  5

                           COMPARE PROCEDURE
                  COMPARISON OF NEW WITH DATABASE.MASTER

                          COMPARISON SUMMARY

         DATASET           OBS  VARIABLES       COMPARISON RESULTS
         NEW                 1          5       EQUAL              1
         DATABASE.MASTER    16          5       UNEQUAL            2
         MATCHED             1          5       TOTAL              3


                          COMPARISONS EQUAL

                      VARIABLE  TYPE  LEN

                      A         NUM     8


                         COMPARISONS UNEQUAL

                      VARIABLE  TYPE  LEN  NDIF

                      B         NUM     8     1
                      C         NUM     8     1
```

```
COMPARISON OF UPDATE DATA WITH DATA BASE BEFORE UPDATE                    6

                        COMPARE PROCEDURE
                COMPARISON OF NEW WITH DATABASE.MASTER

                                    ||  VARIABLE    COMPARE
      YEAR         QTR              ||         B          B    DIFF.     % DIFF
  ----------   ----------           ||  ----------  ---------  ---------  --------
                                    ||
   1982.00      2.00000             ||     999.00     342.99    -656.01  -65.66682

                                    ||  VARIABLE    COMPARE
      YEAR         QTR              ||         C          C    DIFF.     % DIFF
  ----------   ----------           ||  ----------  ---------  ---------  --------
                                    ||
   1982.00      2.00000             ||   66.00000   17.88796   -48.11204 -72.89703
```

Comparing Variables within the Same Data Set: Example 3

Given a data set with variables X, Y, and Z, find the differences between X and Y, X and Z, and Y and Z, and print summary statistics for each comparison.

```
PROC COMPARE STATS;
   VAR  X X Y;
   WITH  Y Z Z;
```

Output 28.3 Comparing Variables within the Same Data Set

```
                                                                          7
                        COMPARE PROCEDURE
                COMPARISONS OF VARIABLES IN DATA1

                        COMPARISON SUMMARY

DATASET           OBS  VARIABLES        COMPARISON RESULTS
DATA1              10          4        EQUAL               0
                                        UNEQUAL             3
                                        TOTAL               3



                    ALL COMPARISONS UNEQUAL

      VARIABLE  TYPE  LEN   COMPARE   LEN  NDIF

      X         NUM     8   Y           8    10
      X         NUM     8   Z           8     2
      Y         NUM     8   Z           8    10
```

COMPARE PROCEDURE
COMPARISONS OF VARIABLES IN DATA1

OBS	VARIABLE X	COMPARE Y	DIFF.	% DIFF
1	1.18496	2.49679	1.31183	110.71
2	2.97009	2.30604	-0.66405	-22.35778
3	3.39982	3.94087	0.54104	15.91382
4	4.25940	5.69862	1.43923	33.78940
5	5.92160	6.47501	0.55341	9.34558
6	6.96928	6.61771	-0.35157	-5.04451
7	7.54298	6.74075	-0.80223	-10.63547
8	8.53169	8.23654	-0.29515	-3.45945
9	9.04979	9.68850	0.63871	7.05773
10	10.06657	8.21800	-1.84856	-18.36340
NDIF	10	(100%)		
R, RSQ	0.943	0.888		
N	10	10	10	10
MEAN	5.98962	6.04188	0.05227	11.69523
STD	2.93879	2.47013	1.02655	38.59744
MAX	10.06657	9.68850	1.43923	110.71
MIN	1.18496	2.30604	-1.84856	-22.35778
STDERR	0.92933	0.78112	0.32462	12.20558
T	6.445	7.735	0.161	0.958
PROB>|T|	0.0001	0.0001	0.8756	0.3630

OBS	VARIABLE X	COMPARE Z	DIFF.	% DIFF
3	3.39982	5.44181	2.04198	60.06137
4	4.25940	5.16413	0.90473	21.24074
NDIF	2	(20%)		
R, RSQ	0.974	0.948		
N	10	10	10	10
MEAN	5.98962	6.28429	0.29467	8.13021
STD	2.93879	2.75159	0.67659	19.42946
MAX	10.06657	10.06657	2.04198	60.06137
MIN	1.18496	1.18496	0	0
STDERR	0.92933	0.87013	0.21396	6.14413
T	6.445	7.222	1.377	1.323
PROB>|T|	0.0001	0.0001	0.2017	0.2184

```
                        COMPARE PROCEDURE                                    9
                  COMPARISONS OF VARIABLES IN DATA1
```

OBS	VARIABLE Y	COMPARE Z	DIFF.	% DIFF
1	2.49679	1.18496	-1.31183	-52.54060
2	2.30604	2.97009	0.66405	28.79591
3	3.94087	5.44181	1.50094	38.08653
4	5.69862	5.16413	-0.53450	-9.37941
5	6.47501	5.92160	-0.55341	-8.54683
6	6.61771	6.96928	0.35157	5.31250
7	6.74075	7.54298	0.80223	11.90122
8	8.23654	8.53169	0.29515	3.58342
9	9.68850	9.04979	-0.63871	-6.59245
10	8.21800	10.06657	1.84856	22.49408
NDIF	10	(100%)		
R, RSQ	0.931	0.868		
N	10	10	10	10
MEAN	6.04188	6.28429	0.24241	3.31144
STD	2.47013	2.75159	1.00545	25.43798
MAX	9.68850	10.06657	1.84856	38.08653
MIN	2.30604	1.18496	-1.31183	-52.54060
STDERR	0.78112	0.87013	0.31795	8.04419
T	7.735	7.222	0.762	0.412
PROB>|T|	0.0001	0.0001	0.4653	0.6902

Plotting the Percent Differences of Variables: Example 4

Given yearly data sets ONE and TWO, plot the percent differences of the variables X1-X10 against time. Note: output for this example has been omitted.

```
PROC COMPARE DATA=ONE COMPARE=TWO OUT=THREE OUTPERCENT NOPRINT;
   VAR X1-X10;
   ID  YEAR;
PROC PLOT DATA=THREE;  PLOT ( X1-X10 ) * YEAR;
```

This could also have been done with a DATA step (using ARRAYs and a RENAME= data set option), but using PROC COMPARE is easier.

Chapter 29

The CONTENTS Procedure

Operating systems: All

ABSTRACT

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

INTRODUCTION

Refer to chapters on the DATA step and the PROC step and especially to the chapter on "SAS Files" to familiarize yourself with the basic features of SAS libraries and the types of files that can reside in them. Also, refer to these chapters if you need help with the terminology used in this procedure description.

PROC CONTENTS provides information about SAS data libraries and about individual SAS files in a SAS library.[1] It is particularly useful for documenting permanent SAS files. The output from PROC CONTENTS includes information such as the file's name, size, type, and when it was created.

Specific information pertaining to the physical characteristics of a member depends on whether the file is a SAS data set or another type of SAS file, such as a file containing graphs or formats. Also, the amount and appearance of the information provided by PROC CONTENTS may vary slightly depending on the operating system you are using. Example output is included for each operating system under which the SAS System runs.

SPECIFICATIONS

The PROC statement is the only statement associated with the CONTENTS procedure. Its form is

PROC CONTENTS *options*;

PROC CONTENTS Statement

PROC CONTENTS *options*;

You can specify the following options in the PROC CONTENTS statement. Included with each description is a list of operating systems under which the option can be used.

DATA=*libref.member*
refers to an entire library or to a specific member within a library to be described by CONTENTS. *Libref* is the first level of a two-level SAS name that refers to the SAS library, and *member* is either the name of a SAS data set in the library or the keyword _ALL_.[2]

When you specify the name of a SAS data set, you get a report on that data set. When you specify _ALL_, you get information about all members of the library of the type or types specified by MEMTYPE=. You cannot specify the member name of any SAS file except a SAS data set.

You must specify the DATA= option with _ALL_ as the member name in order to use the MEMTYPE= option below.

For example, to print a description of all the members in the SAS library referenced by the libref STUDY, use the statement:

```
PROC CONTENTS DATA=STUDY._ALL_ MEMTYPE=ALL;
```

To obtain the contents of one SAS data set called HTWT from the same library, use the statement:

```
PROC CONTENTS DATA=STUDY.HTWT;
```

Note that the DATA= specification is affected by the MEMTYPE= option, which is described below. See MEMTYPE= for details.

DIRECTORY
prints the list of members contained in the SAS library specified by DATA=.[3] This option is useful when you are printing the contents of just one file and when you also want to see the directory for the library. (When _ALL_ is specified in the DATA= option of the PROC statement, the directory is automatically printed.)

Operating systems: All

HISTORY
prints the history of a SAS data set.

The history includes the SAS statements and descriptions of other data sets (if any) used to create a data set. In the history, the current SAS data set is called data set #1. If data set #1 was created from other SAS data sets (by a SET, MERGE, or UPDATE statement), those data sets are called #2, #3, and so on, of generation 1. This process continues until all the history information of all the generations stored is printed. The number of generations stored depends on the value of the SAS system option GEN.

Operating systems: CMS, OS, VM/PC, and VSE

MEMTYPE=
MTYPE=
MT=

specifies the types of members that are available for processing. You must specify _ALL_ as the member name in the DATA= option to use the MEMTYPE= option. The following values can be specified:

ALL	all of the following types
CAT	catalog
DATA	SAS data set
FORMATC	permanent character formats[4]
FORMATN	permanent numeric formats[5]
GCAT	graphics catalog
IMSWK	IML workspace
MODEL	model file

If MEMTYPE= is not specified, the default value is DATA. If you have files with the same name but different types, such as a type DATA file STUDY.INFO and a type FORMATC file STUDY.INFO, the following statement

```
PROC CONTENTS DATA=STUDY.INFO;
```

produces a report on the DATA type only.

You can specify more than one type by enclosing the list of types in parentheses. For example,

```
PROC CONTENTS DATA=STUDY._ALL_ MEMTYPE=(DATA GCAT);
```

produces descriptions of all DATA and GCAT types in the library.

Operating systems: All

NODS

suppresses printing the contents of individual files when _ALL_ is specified in the DATA= option. Only the SAS library directory is printed.

Operating systems: All

NOPRINT

specifies that you do not want the CONTENTS procedure output to be printed. The OUT= option should also be specified when this option is used.

Operating systems: All

NOSOURCE

suppresses printing of the SAS statements used to create a SAS data set. This option is for SAS data sets (DATA type) only.

Operating systems: CMS, OS, VM/PC, and VSE

OUT=*SASdataset*

gives the name of an output SAS data set. The output SAS data set contains the same information that is given in the variable description section in the printed output of CONTENTS. See the **Output** section for more details on this data set. This option is for SAS data sets (DATA type) only.

Operating systems: All

POSITION
prints a second list of the variable names in the order of their position in the data set. By default, the variables are listed in alphabetical order. This option is for SAS data sets (DATA type) only.

Operating systems: All

SHORT
prints only the list of variable names in the SAS data set. This option is for SAS data sets (DATA type) only.

Operating systems: All

DETAILS

Output Data Sets

The output SAS data set optionally produced by PROC CONTENTS contains information on the variables in the individual SAS data set or for all data sets in the library based on the DATA= value. The variables in the output data set are

FORMAT	variable format (blank if none given)
FORMATD	number of decimals for format (0 if none given)
FORMATL	format width (0 if none given)
INFORMAT	variable informat (blank if none given)
INFORMD	number of decimals for informat (0 if none given)
INFORML	informat width (0 if none given)
JUST	justification (0=left, 1=right)
LABEL	variable label (blank if none given)
LENGTH	variable length
LIBNAME	libref used for the data library
MEMLABEL	data set label for this member (blank if no label)
MEMNAME	member in which the variable is located
NAME	variable name
NOBS	number of observations in data set (missing if tape fmt)
NPOS	relative position in the data set input buffer
TYPE	1=numeric 2=character
TYPEMEM	TYPE= value for this member (blank if no TYPE= value)
VARNUM	variable number in data set

For each SAS data set reported (identified by MEMNAME), the variable names are sorted. Note that they are sorted so that the values X1, X2, and X10 come out in that order, not the true collating sequence of X1, X10, X2. Therefore, if you want to use a BY statement on MEMNAME in subsequent steps, you should run a PROC SORT on the output data set first.

Printed Output

The amount of information and the appearance of the output produced by CONTENTS differ somewhat depending on the operating system you are using,

the type of member being described, and the options you specify on the PROC CONTENTS statement.

For SAS data sets, PROC CONTENTS always produces the following:

- the number of OBSERVATIONS in the data set
- data set LABEL, if one exists
- an ALPHABETIC LIST OF VARIABLES. This list includes:
  1. #, the position of the variable in the data set
  2. the VARIABLE name
  3. the TYPE of the variable (character or numeric)
  4. the variable's LENGTH
  5. the starting POSITION of the variable in the observation
  6. the variable's FORMAT and INFORMAT (if any)
  7. the variable's LABEL, if there is one.

Additional information is also produced under some operating systems. The examples in this chapter include a description of the information produced under the different operating systems.[6]

EXAMPLES

PROC CONTENTS under AOS/VS, PRIMOS, and VMS: Example 1

PROC CONTENTS prints the SAS data library name and member names for SAS files. If the file is a SAS data set, the data shown in the example below and described under **Printed Output** is also produced.

Output 29.1 PROC CONTENTS Output under AOS/VS, PRIMOS, and VMS

```
                                        PROC CONTENTS OUTPUT

                                         CONTENTS PROCEDURE
DSNAME=SASABC.MISC.SASDATA                                          GENERATED BY PROC COPY
DATA SET LABEL: old graph data
NUMBER OF OBSERVATIONS: 45                              NUMBER OF VARIABLES: 5
MEMTYPE:   DATA

                                  ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  COLOR     NUM       3        4                           COLOR FOR THIS OBSERVATION
  2  LENGTH    NUM       3        7                           LENGTH OF THIS OBSERVATION
  4  PART1     CHAR    200       13                           FIRST HALF OF DATA
  5  PART2     CHAR    200      213                           LAST HALF OF DATA
  3  PICTURE   NUM       3       10                           PICTURE NUMBER
```

PROC CONTENTS under CMS and VM/PC: Example 2

Under CMS, PROC CONTENTS prints a file description that includes:

- CREATED BY, the userid that created the data set.
- FILEID, the file name and file type.
- CPUID, the CPU serial number.
- the date and time of creation.
- the release of SAS that created the data set.
- how the data were generated (that is, by a DATA step or by a PROC step).
- BLKSIZE, the size of the blocks written on the tape or disk device that contain observations for this individual SAS file. (For FBA disks it is the logical block size.)

- LRECL, the logical record (observation) length of this individual SAS file.

If the file is a SAS data set, SOURCE STATEMENTS, the SAS statements that generated the data set, are produced in addition to the data described under **Printed Output**.

The first PROC CONTENTS statement in this example shows how the procedure, under CMS, prints the contents of all SAS data sets in a SAS library. The NOSOURCE option suppresses the printing of source statements. The second CONTENTS statement prints one SAS data set's description and the data library directory.

```
PROC CONTENTS DATA=TRAIN._ALL_ NOSOURCE;
PROC CONTENTS DATA=TRAIN.MARKLIN DIRECTORY;
```

Output 29.2 PROC CONTENTS Output under CMS and VM/PC

```
                                                    24 JAN 1985 09:42

CONTENTS PROCEDURE
SAS DATA SET DIRECTORY


NAME                        #OBS


HOUSE                        15
SCREEN              CAT


CONTENTS OF SAS DATA SET INVNTRY.HOUSE

CREATED BY CMS USERID SASABC   ON CPUID xx-xxxx-xxxxxx
AT 11:25 WEDNESDAY, JANUARY 23, 1985   BY SAS RELEASE 5.xx
FILE= INVNTRY  HOUSE   GENERATED BY PROC FSEDIT
NUMBER OF OBSERVATIONS: 15   NUMBER OF VARIABLES: 5

----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
   #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
   4  CATEGORY  CHAR      4       83                           Item Category
   3  DESCRIP   CHAR     50       33  CHAR50.                  Item Descripti
   2  ITEM      CHAR     25        8  CHAR25.
   1  ROOM      CHAR      4        4
   5  VALUE     NUM       7       87  DOLLAR7.


CONTENTS OF SAS DATA SET INVNTRY.SCREEN


CREATED BY CMS USERID SASABC   ON CPUID xx-xxxx-xxxxxx
AT 11:25 WEDNESDAY, JANUARY 23, 1985   BY SAS RELEASE 5.xx
FILE= INVNTRY  5SCREEN   GENERATED BY PROC FSEDIT
UTILITY FILE:   SCREEN.CAT
```

PROC CONTENTS under OS: Example 3

PROC CONTENTS prints these PHYSICAL CHARACTERISTICS of the SAS library:

- DSNAME, the physical data set or file where the library is stored by the operating system
- the UNIT value for the device on which the library is stored
- VOL=SER, the volume serial of the device
- DISP, disposition of the library
- the DEVICE type
- MAX BLKSIZE, the maximum length of a standard physical block on the device
- CREATED, the date that the SAS library was created
- TRACKS ALLOCATED, the number of tracks and EXTENTS currently allocated

- the COST of storing the data library on- and off-line (cost information may or may not appear, depending on your installation).

When the DIRECTORY option is specified or when _ALL_ is specified in the DATA= option, a SAS DATA LIBRARY DIRECTORY is printed that includes:

- the NAME of each SAS file contained in the SAS library
- the number of TRACKS used
- the number of SUBEXTENTS that the file occupies
- TOTAL TRACKS USED, the sum of the TRACKS each SAS file uses
- HIGH TRACK USED, the highest-numbered track used.

PROC CONTENTS also prints a description of the SAS files in the library, as requested. The following information is printed:

- the number of TRACKS USED for the SAS file.
- the number of SUBEXTENTS (a subextent is space allocated to the data set on contiguous tracks) that the file occupies.
- CREATED BY JOB, the jobname of the job that created the file.
- the date and time of creation.
- the release of SAS that created the file.
- DSNAME, the name of the SAS library containing the file.
- BLKSIZE, the size of the blocks written on the tape or disk device that contain observations for this individual SAS file. (For FBA disks it is the logical block size.)
- LRECL, the logical record (observation) length of this individual SAS data set.

If the file being described is a SAS data set, PROC CONTENTS produces, in addition to the data listed previously in **Printed Output**:

- OBSERVATIONS PER TRACK, the number of observations that will fit on one track
- how the data were generated (that is, by a DATA step or by a PROC step).

Also printed (unless NOSOURCE is specified) are

- SOURCE STATEMENTS, SAS statements that generated the data set.

The first PROC CONTENTS statement in this example prints the contents of all SAS data sets in a SAS library. The NOSOURCE option suppresses the printing of source statements. The second CONTENTS statement prints one data set description and the data library directory.

```
PROC CONTENTS DATA=MISC._ALL_ NOSOURCE;
PROC CONTENTS DATA=MISC.ECONOMIC DIRECTORY;
```

Output 29.3 PROC CONTENTS Output under OS

```
                                    PROC CONTENTS OUTPUT

                                    CONTENTS PROCEDURE
                                PHYSICAL CHARACTERISTICS OF THE OS DATA SET


DSNAME= SASABC.MISC.SASDATA   UNIT=DISK   VOL=SER=SAS123.   DISP=SHR   DEVICE=3380 DISK   MAX BLKSIZE=32760 BYTES
CREATED ON OCTOBER 5, 1984   10 TRACKS ALLOCATED IN 1 EXTENT(S)    COST IF DATA IS STORED ONLINE= $0.1200/DAY
COST IF DATA IS STORED OFFLINE= $0.0400/DAY


                                    SAS DATA LIBRARY DIRECTORY

                                NAME        MEMTYPE     #OBS        TRACKS      EXTENTS

                                GRAPH       DATA        45             1           1
                                WHISPER     DATA        2              1           1
                                BUILD1      CAT                        1           1
                                COMBINE     GCAT                       1           1
                                NEW         CAT                        1           1
                                OUT         CAT                        1           1
                                TDS         CAT                        1           1
                                TEMPLATE    CAT                        1           1

                                   TOTAL TRACKS USED =     9
                                   HIGH  TRACKS USED =    10


                                    CONTENTS OF SAS MEMBER IN.BUILD1

CREATED BY TSO USERID SASDEF   ON CPUID 12-3456-789990   AT 11:24 THURSDAY, OCTOBER 4, 1984   BY SAS RELEASE 5.03
DSNAME=SASDEF.SAS.GREPLAY3   OBSERVATIONS PER TRACK =100   BLKSIZE=879   LRECL=175   GENERATED BY PROC BUILD
MEMTYPE:   CAT


                                   CONTENTS OF SAS MEMBER IN.COMBINE

CREATED BY TSO USERID SASDEF   ON CPUID 12-3456-789990   AT 10:18 WEDNESDAY, MAY 23, 1984   BY SAS RELEASE 5.00
DSNAME=SASDEF.SAS.GREPLAY3   OBSERVATIONS PER TRACK =20   BLKSIZE=632   LRECL=628   GENERATED BY PROC GREPLAY
MEMTYPE:   GCAT
```

```
                                    PROC CONTENTS OUTPUT

                                   CONTENTS OF SAS MEMBER IN.GRAPH

CREATED BY TSO USERID SASABC   ON CPUID 12-3456-789990   AT 14:50 FRIDAY, DECEMBER 14, 1984   BY SAS RELEASE 5.05
DSNAME=SASABC.MISC.SASDATA   OBSERVATIONS PER TRACK =111   BLKSIZE=15285   LRECL=413   GENERATED BY PROC COPY
DATA SET LABEL: old graph data   DATA SET TYPE: GRAPHICS   NUMBER OF OBSERVATIONS: 45   NUMBER OF VARIABLES: 5
MEMTYPE:   DATA

                                  ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  COLOR     NUM       3        4                           COLOR FOR THIS OBSERVATION
  2  LENGTH    NUM       3        7                           LENGTH OF THIS OBSERVATION
  4  PART1     CHAR    200       13                           FIRST HALF OF DATA
  5  PART2     CHAR    200      213                           LAST HALF OF DATA
  3  PICTURE   NUM       3       10                           PICTURE NUMBER


                                    CONTENTS OF SAS MEMBER IN.NEW

CREATED BY TSO USERID SASDEF   ON CPUID 12-3456-789990   AT 14:56 MONDAY, AUGUST 20, 1984   BY SAS RELEASE 5.00
DSNAME=SASDEF.SAS.GREPLAY3   OBSERVATIONS PER TRACK =200   BLKSIZE=964   LRECL=96   GENERATED BY PROC GREPLAY
MEMTYPE:   CAT
```

(continued on next page)

(continued from previous page)

```
                                CONTENTS OF SAS MEMBER IN.OUT

CREATED BY TSO USERID SASDEF   ON CPUID 12-3456-789990   AT 13:27 WEDNESDAY, AUGUST 15, 1984   BY SAS RELEASE 5.00
DSNAME=SASDEF.SAS.GREPLAY3   OBSERVATIONS PER TRACK =200   BLKSIZE=964   LRECL=96   GENERATED BY PROC GREPLAY
MEMTYPE:   CAT


                                CONTENTS OF SAS MEMBER IN.TEMPLATE

CREATED BY TSO USERID SASDEF   ON CPUID 12-3456-789990   AT 14:45 MONDAY, AUGUST 20, 1984   BY SAS RELEASE 5.00
DSNAME=SASDEF.SAS.GREPLAY3   OBSERVATIONS PER TRACK =200   BLKSIZE=964   LRECL=96   GENERATED BY PROC GREPLAY
MEMTYPE:   CAT
```

```
                                PROC CONTENTS OUTPUT

                                CONTENTS OF SAS MEMBER IN.WHISPER

CREATED BY TSO USERID SASABC   ON CPUID 12-3456-789990   AT 16:27 FRIDAY, DECEMBER 28, 1984   BY SAS RELEASE 5.05
DSNAME=SASCWM.MISC.SASDATA   OBSERVATIONS PER TRACK =1656   BLKSIZE=15460   LRECL=28   GENERATED BY DATA
DATA SET LABEL: old graph data   DATA SET TYPE: GRAPHICS   NUMBER OF OBSERVATIONS: 2   NUMBER OF VARIABLES: 3
MEMTYPE:   DATA

                                ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
  #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
  1  X         NUM       8        4
  2  Y         NUM       8       12
  3  Z         NUM       8       20


                        ---------- ---------- SOURCE RECORDS ----------------------------
                        | data wally.secret (protect=XXXXX);                            |
                        | input x y z;                                                  |
                        | cards;                                                        |
                        ----------------------------------------------------------------
```

PROC CONTENTS under VSE

Under VSE, PROC CONTENTS prints these PHYSICAL CHARACTERISTICS of the SAS library:

- FILENAME, the filename (libref) referencing the JCL statement defining the data set.
- VOL=, the volume of the device.
- STARTING TRACK (STARTING BLOCK for FBA disks), the track on which the file starts.
- NUMBER OF TRACKS (NUMBER OF BLOCKS for FBA disks) allocated to the file.
- the DEVICE type.
- MAX BLKSIZE, the maximum length of a logical block on the device.
- TRACKS ALLOCATED (BLOCKS ALLOCATED for FBA disks), the number of tracks and extents currently allocated.
- for FBA disks, ONE TRACK=ONE CONTROL INTERVAL= *n* BLOCKS, where *n* indicates the number of blocks. Note that for FBA disks, a track corresponds to a control interval.
- PERCENT OF FILE USED.
- the cost of storing the data library on- and off-line (cost information may or may not appear, depending on your installation).

PROC CONTENTS also prints a description of the SAS files in the library, as requested. The following information is printed:

- the number of TRACKS USED for the SAS file.

- the number of SUBEXTENTS that the file occupies. (A subextent is space allocated to the data set on contiguous tracks.)
- CREATED BY JOB, the jobname of the job that created the file.
- the date and time of creation.
- the release of SAS that created the file.
- DSNAME, the name of the SAS library containing the file.
- BLKSIZE, the size of the blocks written on the tape or disk device that contain observations for this individual SAS file. (For FBA disks it is the logical block size.)
- LRECL, the logical record (observation) length of this individual SAS file.

If the file being described is a SAS data set, PROC CONTENTS also produces, in addition to the data previously described in **Printed Output**:

- OBSERVATIONS PER TRACK, the number of observations that will fit on one track.
- how the data were generated, that is, by a DATA step or by a PROC step.

Also printed (unless NOSOURCE is specified) are

- SOURCE STATEMENTS, SAS statements that generated the data set.

The first PROC CONTENTS statement in the following statements print the contents of all SAS data sets in a SAS library. The NOSOURCE option suppresses the printing of source statements. The second CONTENTS statement prints one data set's description and the data library directory. Note that the data library is on an FBA disk.

```
PROC CONTENTS DATA=UFBA009._ALL_ NOSOURCE;
PROC CONTENTS DATA=UFBA009.FORECAST DIRECTORY;
```

Output from PROC CONTENTS under VSE is very similar to output under OS. See **Example 3**, which shows output produced under the OS operating system.

NOTES

1. **AOS/VS, CMS, PRIMOS, VM/PC, and VMS**: A library is a logical entity that relates files rather than a physical entity.

   OS and VSE: A library is a physical entity that includes a directory and encompasses all of the members in the library.

2. **CMS, OS, VM/PC, and VSE**: PROC CONTENTS will not display password protected members when _ALL_ is specified as the member name. You must specify the password of a password-protected SAS data set when you specify the member name in the DATA= option.

3. **OS and VSE**: The characteristics of the SAS data library are also reported, including its size and creation date.

4. **OS, CMS, VM/PC, and VSE**: This type is not available.

5. See note 4.

6. **CMS, OS, and VSE**: For SAS data sets stored on tape, CONTENTS shows the file sequence number and the actual density (bpi) at which the tape was recorded. For standard-labeled tapes, PROC CONTENTS gives the date that the tape was created.

Chapter 30

The CONVERT Procedure

Operating systems: OS, CMS, VM/PC, and VSE

ABSTRACT

The CONVERT procedure converts BMDP®, DATA-TEXT, OSIRIS, SPSS® system files, and SAS data sets from Release 72 to SAS data sets that are compatible with the current release of the SAS System.*

INTRODUCTION

CONVERT produces one output data set, but no printed output. The new data set contains the same information as the input system file; exceptions are noted below under **Output Data Sets.**

The procedure converts system files from these packages:

- BMDP save files through and including the 1983 version of BMDP
- SPSS save files up to and including release 9
- DATA-TEXT files up to and including Version 3.4

* The BMDP Statistical Software Package is a registered trademark of BMDP Statistical Software Inc. SPSS is a registered trademark of SPSS Inc.

- OSIRIS files through and including OSIRIS IV (hierarchical file structures are not supported)
- SAS data sets from Release 72. (SAS Release 72 is a precursor to the current SAS product. It is no longer supported as a product, but files created by this release that need to be converted to the current structure of SAS data sets may still exist.)

Except for SAS Release 72, these packages are maintained by other companies. Changes, therefore, may be made that make the system files incompatible with the current version of CONVERT. For example, at the present time, the CONVERT procedure does not support SPSS-X files. SAS Institute cannot be responsible for upgrading CONVERT to support changes to the packages listed above; however, attempts will be made to do so as necessary with each new version of the SAS System.

Information associated with each package is given in the **Usage Notes** section at the end of the procedure description.

SPECIFICATIONS

PROC CONVERT *options*;

Usually only the PROC CONVERT statement is used, although data set attributes can be controlled by specifying the DROP=, KEEP=, or RENAME= data set options with the OUT= option of this procedure. (Refer to "SAS Files" for more information on these data set options.) You can also use LABEL and FORMAT statements following the PROC statement.

PROC CONVERT Statement

PROC CONVERT *options*;

The options below can appear in the PROC CONVERT statement. Only one of the five options specifying a system file (BMDP, DATATEXT, OSIRIS, SAS72, or SPSS) can be given.

In the descriptions below, fileref is the symbolic name that you associate with a permanent file name in the control language of your operating system. You then use the fileref in your program to reference the permanent file name.[1]

BMDP=*fileref* specifies the *fileref* of a data set containing a BMDP save file. The first save file in the data set is converted. If you have more than one save file in the data set, you can use two additional options in parentheses after the *fileref*. The CODE= option lets you specify the CODE of the save file you want, and the CONTENT= option lets you give the save file's content. For example, if a file CODE=JUDGES had a CONTENT of DATA, you could use this statement:

```
PROC CONVERT BMDP=SAVE(CODE=JUDGES CONTENT=DATA);
```

Operating systems: OS, CMS, and VSE

DATATEXT=*fileref* specifies a *fileref* for a data set containing a DATA-TEXT file.

Operating system: OS

OSIRIS=*fileref* gives a *fileref* for a data set containing an OSIRIS file. This data set can contain no more than 1024 variables.

You must also include the DICT option, described below, when you use the OSIRIS= option.

Operating system: OS

DICT=*fileref2* gives the *fileref* of a data set containing the dictionary file for the OSIRIS data set. DICT= must be specified if you use the OSIRIS= option.

Operating system: OS

SAS72=*fileref* specifies a *fileref* for a data set containing a SAS Release 72 file. If the Release 72 data set is a member of a partitioned data set, the DD statement must include the member name of the Release 72 data set.

Operating system: OS

SPSS=*fileref* gives a *fileref* for a data set containing an SPSS file. CONVERT processes only SPSS system files; it does not handle SPSS masterfiles. To convert an SPSS masterfile to a SAS data set, you must first convert the masterfile to an SPSS save file, then use PROC CONVERT with the SPSS option. Alphabetic SPSS values are converted to SAS character values properly only if a PRINT FORMATS (A) statement is used when the SPSS file was created.

Operating systems: OS, CMS, and VSE

FIRSTOBS=*n* gives the number of the observation where the conversion is to begin. This allows you to skip over observations at the beginning of the BMDP, DATA-TEXT, OSIRIS, Release 72, or SPSS system file.

Operating systems: OS, CMS, and VSE

OBS=*n* specifies the number of the last observation to be converted. This allows you to exclude observations at the end of the file.

Operating systems: OS, CMS, and VSE

OUT=*SASdataset* names the SAS data set created to hold the converted data. IF OUT= is omitted, the SAS System still creates a data set and automatically names it DATA*n*, just as if you omitted a data set name in a DATA statement. If it is the first such data set in a job or session, SAS names it DATA1 ; the second is DATA2, and so on. If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the data set converted to SAS format is not permanently saved. See "SAS Files" for more information.

Operating systems: OS, CMS, and VSE

DETAILS

Missing Values

If a numeric variable in the input data set has no value or a system missing value, CONVERT assigns it a missing value.

Caution

Since some variable name translation can occur as indicated in the appropriate section, be sure that the translated names will be unique.

Output Data Sets

BMDP output Variable names from the BMDP save file are used in the SAS data set, except that nontrailing blanks and all special characters are converted to underscores in the SAS variable names. The subscript in BMDP variable names, such as x(1), becomes part of the SAS variable name, with the parentheses omitted: X1. Alphabetic BMDP variables become SAS character variables of length 4. Category records from BMDP are not accepted. The current maximum record length is 12000. The BMDP CATEGORY information is copied into the SAS data set history section.

DATA-TEXT output Unsubscripted DATA-TEXT variable names are used without change; the subscript in subscripted variable names becomes part of the name. If necessary, the name is truncated on the right so that the name and subscript are eight characters or less; for example, LOCATION(10) becomes LOCATI10. Variable labels are copied without change, as are numeric variables. DATA-TEXT value labels are not copied. The DATA-TEXT codebook is copied to the SAS data set history section. A maximum of 1000 variables can be converted into a SAS data set.

OSIRIS output For single-response variables, the V1-V9999 name becomes the SAS variable name. For multiple-response variables, the suffix R*n* is added to the variable name, when *n* is the response: for example, V25R1 would be the first response of the multiple-response V25. If the variable after V1000 has 100 or more responses, responses above 99 are eliminated. Numeric variables that OSIRIS stores in character, fixed-point binary, or floating-point binary mode become SAS numeric variables. Alphabetic variables become SAS character variables; any alphabetic variable of length greater than 200 is truncated to 200. The OSIRIS variable description becomes a SAS variable label, and OSIRIS print format information becomes a SAS format. The OSIRIS codebook is copied to the SAS data set history section.

SAS Release 72 output CONVERT copies variable names, variables, and missing values from a SAS Release 72 data set without change.

SPSS output SPSS variable names and variable labels become variable names and labels without change. SPSS alphabetic variables become SAS character variables of length 4. SPSS blank values are converted to SAS missing values. SPSS print formats become SAS formats, and the SPSS default precision of no decimal places becomes part of the variables' formats. The SPSS codebook is copied to the SAS data set history section. SPSS value labels are not copied.

Usage Notes

BMDP:

BMDP Statistical Software Inc.
Suite 202
1964 Westwood Boulevard
Los Angeles, California 90025
213/475-5700

SPSS:

SPSS
Suite 3000
444 North Michigan Avenue
Chicago, Illinois 60611
312/329-2400

DATA-TEXT:

Pro-Systems/Amcor
6561 Gillis Drive
San Jose, California 95120
408/997-1776

OSIRIS:

Institute for Social Research
Box 1248
Ann Arbor, Michigan 48106
313/764-4417

EXAMPLE

BMDP Save File Conversion

In the example below, a BMDP save file called WERNER (in the data set pointed to by the fileref BMDPSAVE) is converted to a SAS data set. PROC BMDP is used to create the save file via BMDP5D. The unit number for the BMDP save file is 4; FT04001 is the corresponding fileref.

The SAS job continues by executing PROC CONVERT; a SAS data set called WERNER2 is created from the BMDP save file called WERNER. PROC CONTENTS shows the descriptor information, and PROC PRINT shows the SAS data set variable values.

Output 30.1 PROC CONTENTS Shows the Descriptor Information

```
                                CONTENTS OF THE CONVERTED BMDP SAVE FILE                                1

                                        CONTENTS PROCEDURE


                                    CONTENTS OF SAS DATA SET WORK.WERNER2


CREATED BY OS JOB SASTEST   ON CPUID XX-XXXX-XXXXXX   AT 14:36 SATURDAY, JANUARY 5, 1985   BY SAS RELEASE 5.XX
DSNAME=AAAAAAAA.BBBBBBB.CCCCC.DDDDDDD.EEEEEEEE   BLKSIZE=19056   LRECL=44   OBSERVATIONS PER TRACK =433
GENERATED BY PROC CONVERT
DATA SET LABEL:   JANUARY  5, 1985      14:36:42   DATA SET TYPE: DATA   NUMBER OF OBSERVATIONS: 12
NUMBER OF VARIABLES: 10

                                ----ALPHABETIC LIST OF VARIABLES AND ATTRIBUTES-----
   #  VARIABLE  TYPE LENGTH POSITION  FORMAT         INFORMAT  LABEL
   2  AGE       NUM       4        8
   7  ALBUMIN   NUM       4       28
   5  BRTHPILL  NUM       4       20
   8  CALCIUM   NUM       4       32
   6  CHOLSTRL  NUM       4       24
   3  HEIGHT    NUM       4       12
   1  ID        NUM       4        4
   9  URICACID  NUM       4       36
  10  USE       NUM       4       40
   4  WEIGHT    NUM       4       16
```

Output 30.2 PROC PRINT Shows the SAS Data Set Variable Values

```
                                PRINTED LISTING OF THE CONVERTED BMDP SAVE FILE                         2

      OBS     ID      AGE    HEIGHT    WEIGHT    BRTHPILL    CHOLSTRL    ALBUMIN    CALCIUM    URICACID    USE

        1    2381     22       67       144          1          200          43         98          54       1
        2    1946     22       64       160          2          600          35          .          72       1
        3    1610     25       62       128          1          243          41        104         33       1
        4    1797     25       68       150          2           50          38         96         30       1
        5     561     19       64       125          1          158          41         99         47       1
        6    2519     19       67       130          2          255          45        105         83       1
        7     225     20       64       118          1          210          39         95         40       1
        8    2420     20       65       119          2          192          38         93         50       1
        9    1649     21       60       107          1          246          42        101         52       1
       10    3108     21       65       135          2          245          34        106         48       1
       11    1375     21       63       100          1          208          38         98         54       1
       12    2936     21       64       120          2          260          47        106         38       1
```

NOTE

1. Each of the examples below shows how to associate the fileref, which in these examples is BMDPFILE, with the permanent file name by using the control language of your operating system. For OS, use the following DD statement in the JCL that executes the program:

```
//BMDPFILE  DD  DSN=bmdp.save.file,DISP=SHR
```

For TSO, you must allocate the file as illustrated below before referencing the fileref in your job:

```
ALLOC FI(BMDPFILE) DA(bmdp.save.file) SHR
```

If you are using CMS, only BMDP and SPSS system files can be converted since DATATEXT, OSIRIS, and Release 72 systems are not supported under CMS. Use a FILEDEF statement to refer to the system file. For example:

```
FILEDEF BMDPFILE DISK BMDP SAVEFILE A
```

Chapter 31

The COPY Procedure

Operating systems: All

ABSTRACT

The COPY procedure copies an entire SAS data library or selected members of the library. The procedure can copy from disk to disk, from disk to tape, from tape to tape, or from tape to disk. COPY is especially useful for creating backup copies on tape. Optional features allow you to limit processing to specific types of library members, and additional options under AOS/VS, PRIMOS, and VMS allow you to process SAS data sets (library members of type DATA) in transport format.

INTRODUCTION

Refer to chapters on the DATA step and PROC step to acquaint yourself with general terminology used in this procedure description. Also see the "SAS Files" chapter for a discussion of the general concept of a SAS data library on your operating system and the different types of files (or members) that can reside in a SAS data library.

The COPY procedure is a general utility for copying SAS data libraries from one device to another or from one file to another. You can copy the entire library or you can select or exclude members of specific types.

You can limit the number of observations copied from SAS data sets by using the system options FIRSTOBS= and OBS= in an OPTIONS statement before the PROC COPY step. These options are ignored if you are copying other types of SAS files since only SAS data sets (library members of DATA type) contain observations.

PROC COPY provides optional features that are operating system dependent, such as the ability to copy passwords for protected members under CMS, OS, VM/PC, and VSE, and the ability to create and read SAS data sets in transport format under AOS/VS, PRIMOS, and VMS.

SPECIFICATIONS

PROC COPY IN=*libref* **OUT=***libref options*;
  SELECT *members ... / options*;
  EXCLUDE *members ... / option*;

PROC COPY Statement

PROC COPY IN=*libref* OUT=*libref options*;

IN=*libref*
INDD=*libref*
: refers to the SAS data library containing members to be copied. IN= must be specified. IN= and INDD= are equivalent.

OUT=*libref*
OUTDD=*libref*
: refers to the SAS data library into which members are copied. OUT= must be specified. OUT= and OUTDD= are equivalent.

options
: specify optional features for the copy operation, some of which may depend on the operating system you are using; therefore, each of the following descriptions includes a list of operating systems under which the option can be used.

EXPORT
: specifies that the output library (OUT=) is to be written in transport format. If you specify EXPORT, no other options can be specified on the PROC COPY statement. See **Reading and Creating Transport Data Libraries** below.

  This option automatically restricts procedure processing to SAS data sets (members of DATA type).

  Operating systems: AOS/VS, PRIMOS, and VMS

IMPORT
: specifies that the input library (IN=) is a SAS library in transport format. If you specify IMPORT, no other options can be specified on the PROC COPY statement. See **Reading and Creating Transport Data Libraries** below.

  Operating systems: AOS/VS, PRIMOS, and VMS

MEMTYPE=*membertype*
MTYPE=
MT=
: specifies one or more types of members that are to be moved or copied. Valid *membertype* values are[1]

ALL	all member types (default)
CAT	full-screen catalogs
DATA	SAS data sets
GCAT	graphics catalogs
IMLWK	IML workspace files
MACRO	MACRO facility files[2]
MODEL	model files

If you specify more than one type, the list must be enclosed in parentheses. For example:

```
PROC COPY IN=DS1 OUT=DS2 MEMTYPE=(DATA CAT);
```

If MEMTYPE= is not specified, all (ALL) types of members in the library are available for processing.[2]

Operating systems: All

MOVE
causes each member of the input library (IN=) to be deleted after it has been successfully copied. For example, the following statement:

```
PROC COPY IN=OLD OUT=NEW MOVE;
```

copies members of the SAS library referenced by OLD into the SAS library referenced by NEW and deletes the OLD library.

Operating systems: All

NOHISTORY
prevents history information from being "aged" for SAS data sets that are copied. This option applies only to SAS data sets (library members of DATA type). If this option is not specified, COPY will age the original history information by one generation for each copied (OUT=) data set.

Operating systems: CMS, OS, VM/PC, and VSE

PROTECT=
PROT=
specifies the default write-password for members to be copied. The write-password, if one exists, must be specified for members that are MOVEd (deleted when copied). For example, the following statement copies and deletes SAS data sets that are password protected:

```
PROC COPY IN=OLD OUT=NEW MOVE MEMTYPE=DATA PROTECT=SECURE;
```

Operating systems: CMS, OS, VM/PC, and VSE

SELECT Statement

If you want to copy only selected members, use the SELECT statement to list the names of the members to be copied. The general form of the SELECT statement is

SELECT *member ... / options;*

where

member	names one or more SAS files (or members) selected from the SAS library specified by IN= in the PROC

statement and copied to the SAS library specified by OUT= in the PROC statement.

There are several ways to specify member names in the SELECT statement, as described in **Specifying Member Names** below.

options can be one or both of the following, depending on the operating system you are using.

MEMTYPE= specifies types of members to be selected. Valid values are the same as those shown above for the MEMTYPE= option in the PROC statement.

Types specified with MEMTYPE= on the SELECT statement override the MEMTYPE= specification on the PROC statement. For example, the following statements

```
PROC COPY IN=ORIG OUT=NEW MEMTYPE=(GCAT DATA);
   SELECT MATH LANG GEO / MEMTYPE=CAT;
```

copy members named MATH, LANG, and GEO of type CAT. No other members are copied.

If you do not specify MEMTYPE= on the SELECT statement, those types specified with MEMTYPE= in the PROC statement are available to be selected. For example,

```
PROC COPY IN=ORIG OUT=NEW MEMTYPE=(GCAT DATA);
   SELECT MATH LANG GEO;
```

copies the members MATH, LANG, and GEO of types GCAT and DATA.

If MEMTYPE is not specified on the PROC statement or the SELECT statement, MEMTYPE=ALL is in effect.

Operating systems: All

PROTECT= specifies the write-password. You must specify the write-password if you are copying protected members when the MOVE option (described above) in effect. The password is needed only if MOVE is specified.

PROTECT= specified in the SELECT statement overrides the default password specified with PROTECT= in the PROC statement. For example,

```
PROC COPY IN=OLD OUT=NEW MOVE PROTECT=GROUP;
   SELECT MEM2 MEM4 / PROTECT=TWO;
   SELECT OTHER;
```

on the first SELECT statement the password TWO overrides the default password GROUP so that the members MEM2 and MEM4 can be moved to the library referenced by NEW. Member OTHER named in the second SELECT statement has the default password GROUP.

Operating systems: CMS, OS, VM/PC, VSE

EXCLUDE Statement

If you want to copy most of the members in a library, use the EXCLUDE statement to specify the names of those to be excluded from the copy operation. The general form of the EXCLUDE statement is

EXCLUDE *member* ... / MEMTYPE=*membertype*;

where

member
names one or more SAS files (or members) to be excluded from the SAS library specified by OUT= in the PROC statement. See **Specifying Member Names** below.

MEMTYPE=*membertype*
specifies the types of members that are to be excluded. Valid values for *membertype* are the same as those shown above for the MEMTYPE= option in the PROC statement.

Any types specified with MEMTYPE= in the EXCLUDE statement must have been previously specified in the PROC statement or a SELECT statement.

Specifying Member Names

As shown in the examples above, you can select or exclude one member by giving the name in the SELECT or EXCLUDE statement. You can select or exclude more than one by giving a list of names. You can also select or exclude an abbreviated list of members. For example, this statement

```
SELECT TABS TEST1-TEST3;
```

selects members TABS, TEST1, TEST2, and TEST3. Also, you can select a group of members whose names begin with the same letter or letters by entering the common letter(s) followed by a colon (:). For example, you can select the four members shown in the previous example and all other members having names that begin with the letter T by specifying

```
SELECT T:;
```

Remember that the MEMTYPE= option affects which types of members are available to be selected. Therefore, if MEMTYPE=ALL is in effect, this example selects all member types having names beginning with the letter T.

You specify members to be EXCLUDEd the same way that you specify those to be SELECTed. 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,

```
EXCLUDE STATS TEAMS1-TEAMS4 RBI:;
```

excludes the members STATS, TEAMS1, TEAMS2, TEAMS3, TEAMS4, and all the members beginning with the letters RBI from the copy operation. Again, keep in mind that all types having the specified names are excluded if MEMTYPE=ALL (the default) is in effect.

Note: only members named in a SELECT statement or selected by default can be excluded. If you attempt to exclude a member that is not eligible for copying, you will get an error message. For example, if you enter the following statements

```
SELECT GRADES1-GRADES3;
EXCLUDE GRADES4-GRADES5;
```

COPY will copy the selected members and issue an error message for the members listed in the EXCLUDE statement.

DETAILS FOR THE CMS, OS, VM/PC, AND VSE OPERATING SYSTEMS

Cautions

If the target library (OUT=) is an existing SAS data library on disk and you are copying a member that has the same name as an existing member, the new member replaces the existing member in the target library. However, if the existing member is password protected, the member being copied must also have a password that matches the existing member's password.

If the target library is an existing SAS data library on tape (or a tape-format SAS data library on disk), you run the risk of creating members with duplicate names and inadvertently deleting members.[3]

You will create members with duplicate names if

- you copy more than one member to a tape that contains other members, and
- the member with the duplicate name is not the first member copied.

If the first member you copy to a tape has the same name as a member already on the tape, the member being copied is written over the old member with that name, and any members that follow on the tape are deleted.

The problems of deletion and duplication arise because of the way tapes are processed. PROC COPY copies members in the order they appear in the library. When you copy members to tape, SAS takes the first member to be copied and scans the tape to see if a member with that name already exists on the tape. If one does, SAS writes the new member over the old member and deletes any data sets after it on the tape.

Regardless of whether the first member copied is written over an old member or added as the last member on the tape, the tape remains positioned at the end of the first member copied. After copying the first member, SAS does not scan the tape for duplicate member names because the tape would have to be repositioned for each member being copied. If matching names occur after the first member is copied, the target tape will have two members with the same name, but only the first one on the tape will be accessible.

Here is an example of duplication and deletion: You have a tape that contains a SAS data library, with members called WAGE, LABOR, BENEFITS, DEDUCT, CLAIMS, and HOURS, in that order. You are copying a disk SAS data library with the members CLAIMS, LABOR, and HEALTH (in that order) to the tape. PROC COPY scans the tape for a member called CLAIMS. It finds the CLAIMS member on the tape and writes the disk version of CLAIMS over it. The HOURS member is effectively erased. The tape remains positioned at the end of the new version of CLAIMS. COPY then copies the next member, LABOR. Now there are two data sets called LABOR on the tape, and only the old one (the first one on the tape) is accessible to DATA and PROC steps unless you copy the library back to disk. Finally, the HEALTH member is copied.

If you copy the library to disk, COPY writes the first (old) version of LABOR and then replaces it with the second version of LABOR. If you do not want to copy the entire library, specify LABOR in the SELECT statement. COPY selects and copies the first version of LABOR, then selects and writes the second version of LABOR over the first version.

Before you copy members to tape (or to tape-format data libraries on disk), use the CONTENTS procedure to learn the contents of both the input and output data libraries and the position of members in the libraries (see "The CONTENTS Procedure").

Output

A message appears on the SAS log as each member is successfully copied. No printed output is produced.

DETAILS FOR THE AOS/VS, PRIMOS, AND VMS OPERATING SYSTEMS

SAS Data Libraries on Tape

See the "SAS Files" chapter for information on using PROC COPY with SAS data libraries in tape format. Commands used to assign tape drives and mount tapes are documented in the command language reference manual for your operating system. Information about transport data libraries on tape is included below.

Reading and Creating Transport Data Libraries

The ability to create and read transport libraries is an additional feature with the COPY procedure under AOS/VS, PRIMOS, and VMS. A transport library is a SAS library containing one or more SAS data sets that have been converted from standard format to transport format. In this context, *standard format* means AOS/VS, PRIMOS, or VMS standard format. *Transport format* is essentially an intermediate state of being for a SAS data set that is transported from one operating system to another.

PROC COPY converts SAS data sets (members of type DATA) only. If the library referenced by IN= contains other types of SAS files, COPY ignores them. However, COPY should not be used to convert a SAS data set containing non-standard hexidecimal literals to transport format.

Once converted, the SAS data sets in transport format (or the transport data library) can be read with the SAS System running under the AOS/VS, PRIMOS, or VMS operating system using the COPY procedure with the IMPORT option. (Or you can use the TRANSPORT=YES SAS data set option as described in "SAS Files"). The transport data library can also be read by the SAS System running under a CMS, OS, or VSE operating system using the XCOPY procedure with the IMPORT option.

If you are moving a SAS data library between machines linked in a network, you can use PROC COPY to write or read the transport data library on disk; otherwise, you must use PROC COPY to read or write the transport library on tape. Points to consider when processing transport data libraries on disk and on tape are discussed in separate sections below.

When processing a transport data library, you must specify both the IN= and the OUT= options and either the IMPORT or the EXPORT option. No other options are allowed on the PROC statement.

Use the IMPORT option of PROC COPY to read a transport library and convert it to a SAS data library in standard format. For example:

```
PROC COPY IN=libref OUT=libref IMPORT;
```

Use the EXPORT option to convert SAS data sets from standard format to transport format. For example:

```
PROC COPY IN=libref OUT=libref EXPORT;
```

Note: in CMS, OS, VM/PC, and VSE operating environments, the IMPORT and EXPORT options produce a warning message.

You can use the SELECT and EXCLUDE statements with COPY operations involving transport libraries in the same way you use these statements with standard format libraries. For example, the following statements

```
PROC COPY IN=CSTD OUT=TRANS EXPORT;
   SELECT STATS1-STATS3  TEST BC:;
```

select the SAS data sets named STATS1, STATS2, STATS3, TEST, and all SAS data sets having names that begin with the letters BC. These data sets are selected from a SAS library referenced by the libref CSTD and written to a transport data library referenced by TRANS.

Processing transport data libraries on tape If you transport a SAS data library between operating systems that are **not** linked in a network, you use PROC COPY to write a transport data library to tape or read a transport library from tape. In either case you must use the reserved libref IXTAPE in the PROC COPY statement to refer to the transport library. For example, you must specify IN=IXTAPE if you are importing a transport data library on tape. Likewise, specify OUT=IXTAPE if you are exporting a transport data library on tape.

Suppose you want to use PROC COPY to read a transport library from a tape that was created in an OS operating environment and convert it to standard format. You must specify the PROC statement, as follows:

```
PROC COPY IN=IXTAPE OUT=TEMP IMPORT;
```

This statement reads a transport data library from tape, converts it, and creates a SAS data library in standard format referenced by TEMP.

Specify the EXPORT option and OUT=IXTAPE if you are creating a transport data library on tape. For example:

```
PROC COPY IN=FMTSTD OUT=IXTAPE EXPORT;
```

Use the LIBNAME statement (described in "SAS Statements Used Anywhere") to define the libref that refers to the SAS data library in standard format. For the above example, you must issue a LIBNAME statement to associate the libref FMTSTD with the directory containing the SAS data library in standard format. The LIBNAME statement must come before the PROC COPY statement. The following program

```
LIBNAME FMTSTD 'directory1';
PROC COPY IN=FMTSTD OUT=IXTAPE EXPORT;
```

reads all the SAS data sets contained in *directory1*, converts them to transport format, and writes them to tape.

Before using the IXTAPE libref, you must issue control language commands to assign a tape drive to your session. Then, be sure that the appropriate tape is mounted on the assigned tape drive. When SAS encounters the libref IXTAPE, the tape mounted on the assigned tape drive is used to read or write a transport data library. See the *SAS Companion* or SAS Technical Report for your operating system for information on commands used to assign tape drives for your SAS session.

Processing transport data libraries on disk When operating systems are linked in a network, you can use PROC COPY to read or write transport data libraries on disk. No special libref is required for the transport data library on disk. You must use the LIBNAME statement to define the directory for the SAS data library in standard format.

To define the transport data library, issue the appropriate operating system command before you invoke SAS or after you invoke SAS with the X statement, or use the FILENAME statement (described in "SAS Statements Used Anywhere"). If you do not define the transport data library, PROC COPY reads or writes a transport data library in the current default directory. For example, this program

```
LIBNAME FMTSTD 'directory1';
PROC COPY IN=FMTSTD OUT=TRANS EXPORT;
```

reads a SAS data library in standard format from *directory1* and writes a transport data library to a file named TRANS in the current default directory.[4]

Output

A message appears on the SAS log when each data set is successfully copied. No printed output is produced.

EXAMPLES

Copying a SAS Data Library: Example 1

This example copies all members in the SAS data library referenced by the libref MISC to a library referenced by the libref NEW. Since MEMTYPE= is not specified, all types are copied:

```
PROC COPY IN=MISC OUT=NEW;
```

Copying Selected Members of a SAS Data Library: Example 2

In this example, selected members in the library referenced by the libref MISC are copied to a library referenced by the libref NEW. Notice that SAS data sets (DATA types) only are initially available for processing. The MOVE option is in effect, so passwords must be specified if selected members are protected.

```
PROC COPY IN=MISC OUT=NEW MOVE MEMTYPE=DATA;
   SELECT MLIST;
   SELECT GTEST: / MEMTYPE=CAT;
   EXCLUDE GTEST3 / MEMTYPE=CAT;
```

The first SELECT statement copies SAS data set MLIST, which is not password protected. The second SELECT statement copies all members having names beginning with letters GTEST of type CAT except the one named GTEST3 that is listed in the EXCLUDE statement. All of the selected members are copied to the SAS library referenced by NEW and deleted from the library referenced by OLD.

NOTES

1. **AOS/VS, PRIMOS, and VMS**: The following values are also allowed:

FORMATC	permanent character formats
FORMATN	permanent numeric formats

2. **AOS/VS, PRIMOS, and VMS**: If you specify the IMPORT or EXPORT option, only DATA types are available for processing.

3. **VM/PC**: You cannot use tapes in this environment.

4. **VMS**: The extension for an external file is appended, so the file name in the current directory is TRANS.DAT.

Chapter 32

The CORR Procedure

Operating systems: All

ABSTRACT

The CORR procedure computes correlation coefficients between variables, including Pearson product-moment and weighted product-moment correlations. Three nonparametric measures of association (Spearman's rank-order correlation, Kendall's tau-b and Hoeffding's measure of dependence, D) can also be produced. CORR also computes some univariate descriptive statistics.

INTRODUCTION

Correlation measures the closeness of a linear relationship between two variables. If one variable x can be expressed exactly as a linear function of another variable y, then the correlation is 1 or -1, depending on whether the two variables are directly related or inversely related. A correlation of zero between two variables means that each variable has no linear predictive ability for the other. If the values are normally distributed, then a correlation of zero means that the variables are independent of one another.

The true product-moment correlation (Pearson), denoted ρ_{xy}, is defined:

$$\rho_{xy} = \text{cov}(x,y)/\sqrt{(\text{var}(x)\text{var}(y))}$$
$$= E((x - Ex)(y - Ey))/\sqrt{(E(x - Ex)^2E(y - Ey)^2)} \quad .$$

The sample correlation estimates the true correlation. It is computed:

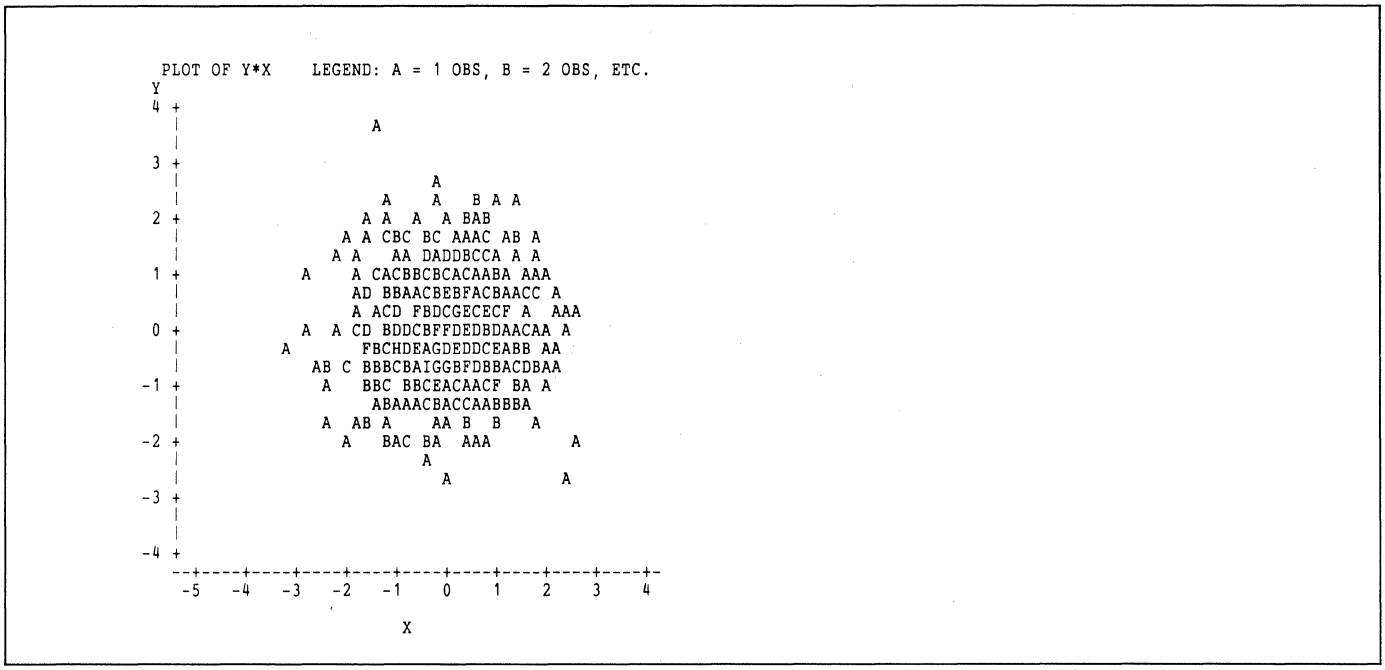
$$r_{xy} = \Sigma(x - \bar{x})(y - \bar{y})/\sqrt{(\Sigma(x - \bar{x})^2\Sigma(y - \bar{y})^2)}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of *x* and *y*.

The full name of this statistic is the *Pearson product-moment correlation*.

For example, **Output 32.1** below is a scatterplot of the variables x and y; its correlation is close to zero, specifically $r=.02$:

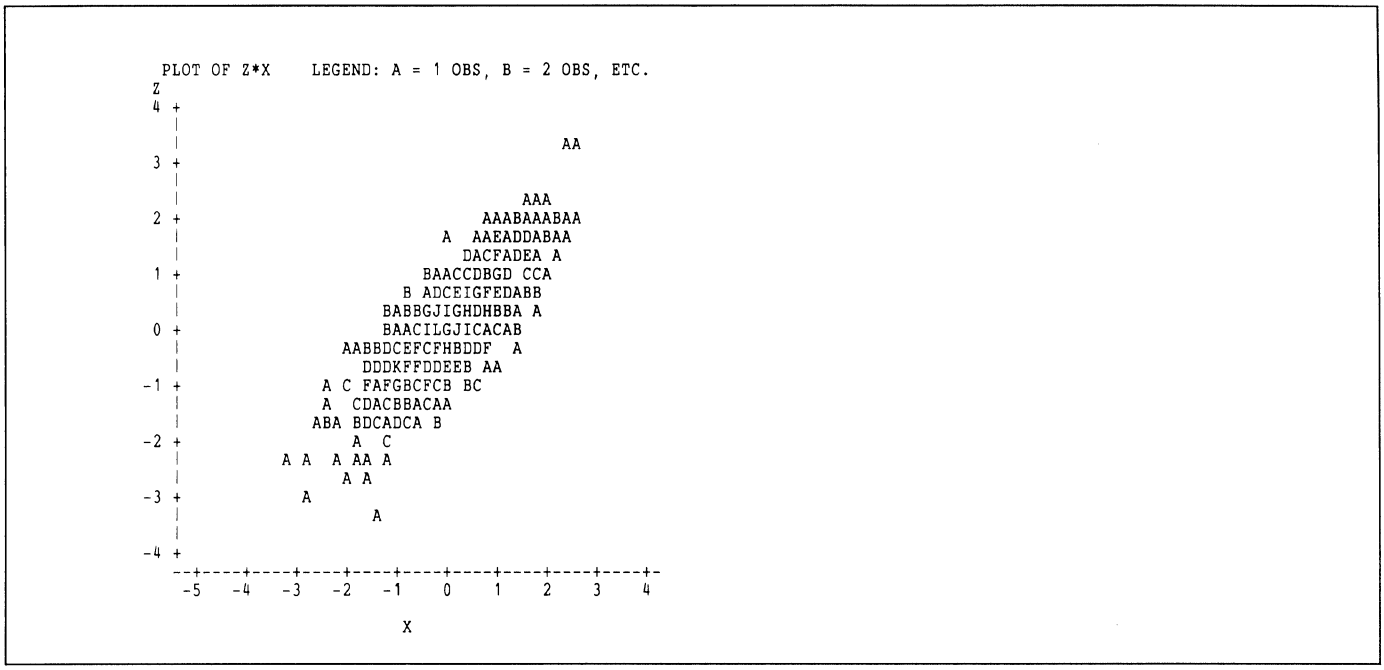
Output 32.1 Scatterplot Showing Variables with a Correlation *r* of .02: PROC CORR



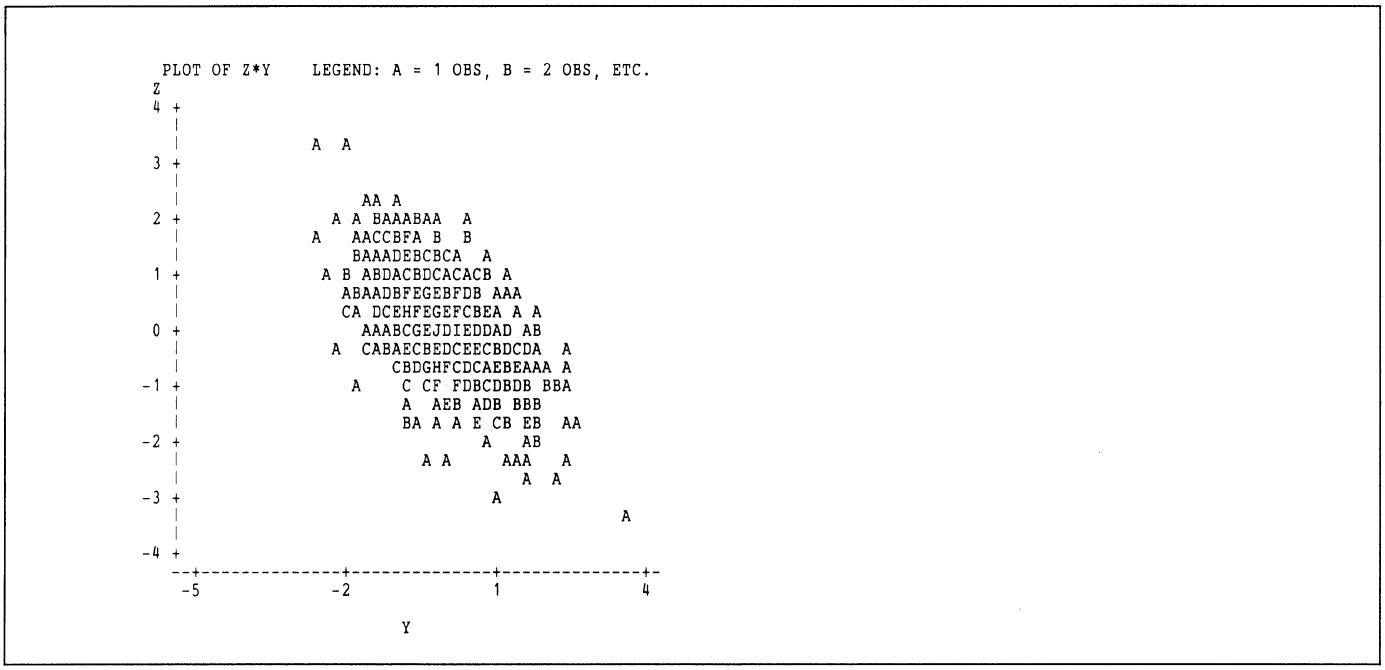
When variables are highly correlated, their points tend to fall on or near a line of fit. In **Output 32.2** *x* and *z* have a correlation *r* of .81.

If the variables are negatively correlated, the line of fit has a negative slope; in other words, the variables are related inversely so that high values of one variable tend to occur with low values of the other. This is the case with *z* and *y*, which have a correlation *r* of −.57 as shown in **Output 32.3**.

Output 32.2 Scatterplot Showing Variables with a Correlation *r* of .81: PROC CORR



Output 32.3 Scatterplot Showing Variables with a Correlation *r* of −.57: PROC CORR



Correlations among a set of variables are statistics that contain sufficient information for computing regressions, partial correlations, canonical correlations, factor patterns, principal component coefficients, and many other statistics.

In addition to Pearson correlations, several other statistics have been proposed to measure association between two continuous variables. Spearman's rank-order correlation coefficient is a nonparametric measure that is calculated as the correlation of the ranks of the data. Kendall's tau-b is a measure calculated from concordances and discordances. Concordance is measured by determining whether values of paired observations vary together (in concord) or differently (in discord). Hoeffding's measure of dependence, D, approximates a weighted sum over observations of chi-square statistics for two-by-two classification tables using each set of *x,y* values as the cut points for the classification. This statistic will detect more general departures from independence than the correlations. (See Hollander and Wolfe 1973, p. 228 for more background on Hoeffding's D.)

SPECIFICATIONS

The following statements can be used in the CORR procedure:

PROC CORR *options*;
  VAR *variables*;
  WITH *variables*;
  WEIGHT *variable*;
  FREQ *variable*;
  BY *variables*;

The PROC CORR statement invokes the procedure. VAR and WITH statements are used to specify variables for the analysis. WEIGHT, FREQ, and BY statements are optional.

There is no fixed order for statements following the PROC CORR statement.

PROC CORR Statement

PROC CORR *options*;

The options that can appear in the PROC CORR statement are given below. If no options are specified, CORR calculates Pearson product-moment correlations and significance probabilities, printing them in a rectangular table along with univariate statistics.

DATA=*SASdataset*
: specifies the name of the SAS data set to be used by PROC CORR. If it is omitted, the most recently created SAS data set is used.

PEARSON
: requests the usual Pearson product-moment correlations. Since these are the default statistics if no options are given, you only need to specify this option if you also are requesting Spearman, Kendall, or Hoeffding statistics. The formula for the Pearson correlation is given above in the **Introduction**.

SPEARMAN
: requests that Spearman coefficients be calculated and printed. These are the correlations of the ranks of the variables. It is appropriate only when both variables lie on an ordinal scale. The formula for the Spearman rank correlation is given under **Formulas** later in this chapter. Range: $-1 \leq r_s \leq 1$. The SPEARMAN option is not valid if you include a WEIGHT statement.

KENDALL
: requests that Kendall's tau-b coefficients be calculated and printed. Kendall's tau-b is based on the number of concordant and discordant pairs of observations and uses a correction for tied pairs (that is, pairs of observations that have equal values of X or equal values of Y). It is appropriate only when both variables lie on an ordinal scale. Range: $-1 \leq$ tau-b ≤ 1. The KENDALL option is not valid if you include a WEIGHT statement. See **Formulas** below for Kendall's tau-b.

HOEFFDING
: requests that Hoeffding's D statistic be calculated and printed. Note that the statistic calculated by CORR is 30 times the usual definition. (See **Formulas** below for Hoeffding's statistic.) This will scale the statistic to range between $-.5$ and 1 with only large positive values indicating dependence. The HOEFFDING option is not valid if you include a WEIGHT statement.

RANK
: requests that the correlation coefficients for each variable be printed in order from highest to lowest in absolute value. When RANK is omitted, the correlations and the other two statistics are printed in a rectangular table defined by variable names at the top and side.

BEST=*n*
: prints the *n* correlations for each variable with the largest absolute values; the coefficients are printed in descending order.

VARDEF=*divisor*
: specifies the divisor to be used in the calculation of variances and covariances. Possible values for *divisor* are N, DF, WEIGHT or WGT, and WDF. VARDEF=N requests that the number of observations be used as the divisor. VARDEF=DF requests that the error degrees of freedom, $n-1$, be used. VARDEF=WEIGHT or WGT requests that the sum of the weights be used. VARDEF=WDF requests that the sum of the weights minus one be used. The default value is DF.

NOSIMPLE
: suppresses printing the simple descriptive statistics for each variable.

NOPRINT
: suppresses printing the correlations. This option is used when the only purpose of using CORR is to create an output data set containing correlations.

NOPROB
: suppresses the printing of significance probabilities associated with the correlations.

NOMISS
: specifies that an observation with a missing value for any variable previously used in the analysis be dropped from all calculations. Otherwise, CORR includes observations that have nonmissing values for pairs. Specifying NOMISS can be much faster and cheaper.

SSCP
: requests that the sums of squares and cross products (SSCP) be printed. If both SSCP and OUTP= (below) are specified, the output data set contains the SSCP matrix as well as the correlation matrix. For the SSCP observations, the _TYPE_ variable's value is SSCP.

The SSCP option is only meaningful with the PEARSON option; do not use with SPEARMAN, KENDALL, or HOEFFDING.

COV
requests that covariances be printed. If COV and OUTP= (below) are both specified, the output data set contains the covariance matrix as well as the correlation matrix. For the observations containing the covariance matrix, the _TYPE_ variable's value is COV. The COV option is only meaningful with the PEARSON option; do not use with SPEARMAN, KENDALL, or HOEFFDING.

NOCORR
specifies that correlations not be output to the output data set. If NOCORR is specified, the data set does not contain correlations, but PROC CORR still makes the data set TYPE=CORR. You can change the data set type to COV or SSCP by using the TYPE= data set option:

```
PROC CORR NOCORR COV OUT=B(TYPE=COV);
```

OUTP=*SASdataset*
requests that CORR create a new SAS data set containing Pearson correlations. The new data set is TYPE=CORR and includes means, standard deviations, simple statistics, and correlation coefficients. Requesting this data set also turns on the PEARSON option. The example at the end of this chaper lists a TYPE=CORR data set. If you want to create a permanent SAS data set, you must specify a two-level name (see "SAS Files," in *SAS User's Guide: Basics* for more information on permanent SAS data sets.) Also see **Output Data Sets** at the end of this chapter for a complete description of the data set's structure.

OUTS=*SASdataset*
requests that CORR create a new SAS data set containing Spearman correlations. Requesting this data set turns on the SPEARMAN option. OUTS= is similar in other respects to OUTP=.

OUTK=*SASdataset*
requests that CORR create a new SAS data set containing Kendall correlations. Requesting this data set turns on the KENDALL option. OUTK= is similar in other respects to OUTP=.

OUTH=*SASdataset*
requests that CORR create a new SAS data set containing the Hoeffding statistic. Requesting this data set turns on the HOEFFDING option. OUTH= is similar in other respects to OUTP=.

VAR Statement

VAR *variables;*

The names of the variables to be correlated are listed in the VAR statement. If you omit the VAR statement, CORR calculates correlations between all the numeric variables in the input data set.

For example, the statements

```
PROC CORR;
   VAR A B C;
```

produce correlation coefficients between three pairs of variables: A and B, A and C, B and C.

WITH Statement

WITH *variables;*

If you want correlations for specific combinations of variables, list the variables in the VAR statement that you want on the top of the printed correlation matrix, and list in the WITH statement variables you want on the side of the correlation matrix.

For example, the statements

```
PROC CORR;
   VAR A B;
   WITH X Y Z;
```

produce correlations for these combinations:

A and X	B and X
A and Y	B and Y
A and Z	B and Z .

WEIGHT Statement

WEIGHT *variable;*

If you want to compute weighted product-moment correlation coefficients, give the name of the weighting variable in a WEIGHT statement. A WEIGHT statement should only be used with Pearson correlations; SPEARMAN, KENDALL, and HOEFFDING options are not valid with a WEIGHT statement.

FREQ Statement

FREQ *variable;*

If one variable in your input data set represents the frequency of occurrence for values of another variable in the observation, include the variable's name in a FREQ statement. CORR then treats the data set as if each observation appeared *n* times, where *n* is the value of the FREQ variable for the observation. The total number of observations is considered equal to the sum of the FREQ variable when CORR calculates significance probabilities. WEIGHT and FREQ statements have similar effects except in calculating degrees of freedom. A FREQ variable can be specified for any correlation, unlike a WEIGHT variable.

BY Statement

BY *variables;*

A BY statement can be used with PROC CORR to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step" in *SAS User's Guide: Basics.*

DETAILS

Missing Values

The default method of handling missing values is to use all the nonmissing pairs of values for each pair of variables. This means that some correlations are computed using more observations than others.

An alternative method is available with the NOMISS option, which uses an observation only if none of the variables is missing.

There are two reasons for specifying NOMISS and avoiding the pairwise default method. First, this method is more efficient computationally. Second, if the correlations are used as input to regression or other statistical procedures, a pairwise-missing correlation matrix leads to several statistical difficulties. Pairwise correlation matrices may not be positive definite. The pattern of missing values may bias the results.

Output Data Sets

Output data sets are requested by the OUTP=, OUTS=, OUTK=, and OUTH= specifications on the PROC CORR statement. OUTS=, OUTK=, and OUTH= produce SAS data sets with Spearman, Kendall, and Hoeffding statistics, respectively. The data sets are marked with the special data set name TYPE=CORR. This type of data set is recognized by many SAS procedures, including REG and FACTOR.

The variables on the data set are

- BY variables
- _TYPE_, identifying the type of observation
- _NAME_, identifying variable names
- the variables listed on the VAR statement.

The observations, as identified by _TYPE_, are

- MEAN, the mean of each variable
- STD, the standard deviation of each variable
- N, the number of nonmissing observations for each variable
- SUMWGT, the sum of the weights for each variable, but only if a WEIGHT statement was specified.

The following kinds of observations are further identified by the combination of the variables _TYPE_ and _NAME_.

- SSCP, the cross products (if the SSCP option is given)
- COV, the covariances (if the COV option is given)
- CORR, a sequence of observations containing correlations.

Note that the SSCP and COV options do not work unless **only** Pearson correlation coefficients are requested.

Formulas

The formula for the Spearman correlation is

$$\theta = \frac{\Sigma(R_i - \bar{R})(S_i - \bar{S})}{\sqrt{(\Sigma(R_i - \bar{R})^2 \Sigma(S_i - \bar{S})^2)}}$$

where R_i is the rank of the ith x value, S_i is the rank of the ith y value, and $\overline{R}$ and $\overline{S}$ are the means of the R_i and S_i values respectively. Averaged ranks are used in the case of ties.

The formula for Kendall's tau-b is

$$\tau = \frac{\Sigma_{i<j}\text{sgn}(x_i - x_j)\ \text{sgn}(y_i - y_j)}{\sqrt{((n(n-1)/2 - \Sigma t_i(t_i - 1))(n(n-1)/2 - \Sigma u_i(u_i - 1)))}}$$

where the t_i (the u_i) are the number of tied x (respectively y) values in the ith group of tied x (respectively y) values, n is the number of observations, and $\text{sgn}(z)=1$ if $z>0$, 0 if $z=0$, and -1 if $z<0$.

The formula for Hoeffding's D statistic, taken from Hoeffding (1948), is

$$D = 30\frac{(n-2)(n-3)S_1 + S_2 - 2(n-2)S_3}{n(n-1)(n-2)(n-3)(n-4)}$$

where $S_1=\Sigma_i(Q_i-1)(Q_i-2)$, $S_2=\Sigma_i(R_i-1)(R_i-2)(S_i-1)\ (S_i-2)$, and $S_3=\Sigma_i(R_i-2)(S_i-2)\ (Q_i-1)$. The R_i and S_i are the ranks of the x and y values as before and the Q_i (sometimes called bivariate ranks) are one plus the number of points which have both x and y values less than the ith points. A point that is tied on its x value or y value, but not both, contributes 1/2 to Q_i if the other value is less than the corresponding value for the ith point, and a point tied on both x and y contributes 1/4 to Q_i.

Computational Method

For the Spearman, Kendall, or Hoeffding correlations, the data are first ranked. The Spearman correlation is then computed on the ranks using the formula for the Pearson correlation. Averaged ranks are used in case of ties. The Kendall correlation is computed using a method similar to Knight (1966). Briefly, the method proceeds in this way: observations are ranked in order according to values of the first variable, the observations are then re-ranked according to values of the second variable, and the number of interchanges of the first variable that occur is noted and used to compute Kendall's tau-b. The Q_i values needed to compute Hoeffding's D are also obtained from this double sorting.

Probability values for the Pearson and Spearman correlations are obtained by treating $(n-2)^{1/2}\ p\ /\ (1-p^2)^{1/2}$ as coming from a t distribution with $n-2$ degrees of freedom, where p is the appropriate correlation. The probability values for the Kendall correlations are obtained by treating $s\ /\ (\text{var}(s))^{1/2}$ as coming from a normal distribution where

$$s = \Sigma_{i<j}\ \text{sgn}(x_i - x_j)\text{sgn}(y_i - y_j)$$

and where x_i are the values of the first variable, y_i are the values of the second variable, and $\text{sgn}(z)= 1$ if $z>0$, 0 if $z=0$, and -1 if $z<0$. The variance of s is computed as:

$$((2n + 5)(n - 1)n - \Sigma_i t_i(t_i - 1)(2t_i + 5) - \Sigma_i u_i(u_i - 1)(2u_i + 5))/18 + (\Sigma_i t_i(t_i - 1)(t_i - 2))(\Sigma u_i(u_i - 1)(u_i - 2))/(9n(n - 1)(n - 2)) + (\Sigma t_i(t_i - 1))(\Sigma u_i(u_i - 1))/(2n(n - 1))$$

where the sums are over tied groups of values, t_i is the number of tied x values, and u_i is the number of tied y values (Noether 1967).

The probability values for Hoeffding's D statistic are computed using the asymptotic distribution computed by Blum, Keifer, and Rosenblatt (1961). The statistic

$$(n - 1)D*\pi**4/60 + \pi**4/72$$

is computed and treated as coming from the asymptotic distribution. Tables for the distribution of D for sample sizes less than 10 are available in Hollander and Wolfe.

Computer Resources

Let K denote the number of correlations to be computed. Then if N, the number of observations, is large, the CPU time required varies as $N*K$ when only Pearson correlations are requested. If Spearman correlations are requested, the CPU time varies as $N*\log N$ times the number of variables involved. If Kendall or Hoeffding statistics are requested, the CPU time scales as $K*N*\log N$.

If N is small and K is large, the CPU time required scales as K for all of the types of correlations.

The only factor limiting the number of variables used by PROC CORR is the required computer resources.

Resources for SPEARMAN, KENDALL, or HOEFFDING options For the most time-efficient processing, the amount of temporary storage needed in bytes is

$$8*V*W*C + N*(28 + 8*V)$$

where

V is the number of variables in the VAR list (or the number of numeric variables if the VAR list is omitted),
W is the number of variables in the WITH list (or is equal to V if there is no WITH list),
C is the number of types of correlations desired,
N is the number of observations, and
NV is V if there is no WITH list and V+W if there is a WITH list.

The minimum temporary storage needed to process the data is

$$8*V*W*C + 52*N \quad .$$

Resources for Pearson correlations alone If only Pearson correlations are requested, the resource use is somewhat different. For time-efficient processing, memory for temporary storage should be at least M, where

$$M = K*8 \text{ if the NOMISS option is specified}$$
$$= K*60 + V*8 \text{ otherwise}$$

where

K = V*W if a WITH list is used
K = $V^2/2$ if not.

If M bytes are not available, CORR must pass through the data several times to collect all the statistics. If this happens, CORR prints a note suggesting a larger memory region.

CPU time varies with D*N*K. D is much smaller if NOMISS is given than if it is not given. Without NOMISS, processing is much faster if most of the observations have no missing values.

Printed Output

For each variable, CORR prints:

1. the VARIABLE name.
2. N, the number of observations having nonmissing values of the variable.
3. the MEAN.
4. STD DEV, the standard deviation.
5. the MEDIAN, if the SPEARMAN or KENDALL option is specified.
6. the MINIMUM value.
7. the MAXIMUM value.
8. the SUM, if only Pearson product-moment correlations are calculated (not shown).

For each pair of variables, CORR prints:

9. the Spearman, Kendall, Hoeffding, and Pearson coefficients requested.
10. N, the number of observations used to calculate the coefficient, under the null hypothesis that the correlation is zero. If a FREQ statement is used, this number is the sum of the frequency variable.
11. PROB> |R|, the significance probability of the correlation or PROB>D, the significance probability of Hoeffding's D. This probability is approximate for Spearman, Kendall, and Hoeffding statistics.

Among its optional features, CORR can:

12. print only the correlations requested with no simple statistics (the NOSIMPLE option).
13. produce the correlations for specific combinations of variables (the WITH statement).
14. produce a specially structured SAS data set containing a correlation matrix, which can be used as input to the FACTOR, SYSLIN, and REG procedures, among others.

EXAMPLE

Three Uses of PROC CORR

In the following example PROC CORR is used three times: (1) to obtain all four kinds of measures of association; (2) to show a rectangular correlation matrix; and (3) to show an output data set with covariances as well as correlations.

```
*---DATA COURTESY A.C. LINNERUD, N.C.STATE UNIV.;
DATA SKINFOLD;
   INPUT CHEST ABDOMEN ARM @@;
   CARDS;
09.0 12.0 3.0  20.0 20.0 7.5  10.0 23.0 6.0  12.0  6.0 5.0
```

```
08.5 15.0 3.0   12.0 17.0 4.0   11.0 13.0 6.0   05.0 14.0 3.0
13.0 19.0 3.0   22.0 20.0 6.0   10.5 12.0 3.5   17.0 15.0 4.5
10.0  7.0 4.0   17.0 28.0 5.5   15.0 15.5 3.0   16.0 11.0 3.0
07.0 13.0 2.5   16.0 18.0 3.0   09.0 12.5 5.0   17.5 18.0 3.0
15.5 28.5 5.0   21.0 27.5 6.0   23.0 24.0 6.5   11.5 15.0 3.0
22.5 20.0 4.5   13.0 14.0 4.0   14.0 21.0 2.5   04.0  3.0 2.0
05.5  8.5 3.0   21.0 13.0 9.0   16.0 11.0 3.0   17.5 15.0 4.5
25.0 35.0 6.5   21.0  6.0 3.5   16.5 17.0 4.0   09.5 11.5 2.5
15.0 19.0 4.0   13.5  6.5 3.5   16.0 15.0 3.0   26.0 38.0 4.0
12.5 20.0 3.0   05.0  7.5 3.5   12.0 15.0 3.5   15.0 13.0 4.5
17.0 19.5 5.0   16.0 20.0 5.5   09.0  4.0 2.0   19.0 12.0 3.0
16.0 17.5 6.0   14.5 14.5 4.0
;

PROC CORR PEARSON SPEARMAN KENDALL HOEFFDING;
   VAR CHEST ABDOMEN ARM;
TITLE 'SPEARMAN''S RHO, KENDALL''S TAU-B, PEARSON''S & HOEFFDING';

PROC CORR DATA=SKINFOLD NOSIMPLE;
   VAR ABDOMEN ARM;
   WITH CHEST;
TITLE 'RECTANGULAR CORRELATION MATRIX';

PROC CORR DATA=SKINFOLD COV OUTP=CORROUT;
   VAR CHEST ABDOMEN ARM;
TITLE 'COVARIANCES AND CORRELATIONS';
PROC PRINT DATA=CORROUT;
TITLE2 'OUTPUT DATASET FROM PROC CORR';
```

Output 32.4 Four Measures of Association

```
                    SPEARMAN'S RHO, KENDALL'S TAU-B, PEARSON'S & HOEFFDING                1
 ❶          ❷                ❸                ❹                ❺                ❻                ❼
VARIABLE        N             MEAN           STD DEV           MEDIAN           MINIMUM          MAXIMUM

CHEST          50       14.41000000       5.25637611       15.00000000        4.00000000       26.00000000
ABDOMEN        50       16.01000000       7.19090270       15.00000000        3.00000000       38.00000000
ARM            50        4.15000000       1.48547388        4.00000000        2.00000000        9.00000000

                                                                                          ❿
                PEARSON CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 50

                                          CHEST     ABDOMEN         ARM

                                CHEST     1.00000   0.61986   0.52973  ❾
                                          0.0000    0.0001    0.0001   ⓫

                                ABDOMEN   0.61986   1.00000   0.42925
                                          0.0001    0.0000    0.0019

                                ARM       0.52973   0.42925   1.00000
                                          0.0001    0.0019    0.0000

                SPEARMAN CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 50

                                          CHEST     ABDOMEN         ARM


                                CHEST     1.00000   0.54825   0.51761
                                          0.0000    0.0001    0.0001

                                ABDOMEN   0.54825   1.00000   0.45687
                                          0.0001    0.0000    0.0009

                                ARM       0.51761   0.45687   1.00000
                                          0.0001    0.0009    0.0000
```

(continued on next page)

(continued from previous page)

KENDALL TAU B CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 50

	CHEST	ABDOMEN	ARM
CHEST	1.00000 0.0000	0.41246 0.0000	0.38974 0.0002
ABDOMEN	0.41246 0.0000	1.00000 0.0000	0.33048 0.0015
ARM	0.38974 0.0002	0.33048 0.0015	1.00000 0.0000

SPEARMAN'S RHO, KENDALL'S TAU-B, PEARSON'S & HOEFFDING 2

HOEFFDING DEPENDENCE COEFFICIENTS / PROB > D UNDER H0:D=0 / N = 50

	CHEST	ABDOMEN	ARM
CHEST	1.00000 0.0000	0.09636 0.0001	0.06797 0.0006
ABDOMEN	0.09636 0.0001	1.00000 0.0000	0.04966 0.0030
ARM	0.06797 0.0006	0.04966 0.0030	1.00000 0.0000

Output 32.5 Producing a Rectangular Correlation Matrix

⓬ RECTANGULAR CORRELATION MATRIX 1

PEARSON CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 50

	ABDOMEN	ARM ⓭
CHEST	0.61986 0.0001	0.52973 0.0001

Output 32.6 Producing an Output Data Set with Covariances and Correlations

COVARIANCES AND CORRELATIONS 1

⓮ COVARIANCE MATRIX

	CHEST	ABDOMEN	ARM
CHEST	27.6295	23.4295	4.13622
ABDOMEN	23.4295	51.7091	4.5852
ARM	4.13622	4.5852	2.20663

COVARIANCES AND CORRELATIONS 2

VARIABLE	N	MEAN	STD DEV	SUM	MINIMUM	MAXIMUM
CHEST	50	14.41000000	5.25637611	720.50000000	4.00000000	26.00000000
ABDOMEN	50	16.01000000	7.19090270	800.50000000	3.00000000	38.00000000
ARM	50	4.15000000	1.48547388	207.50000000	2.00000000	9.00000000

PEARSON CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 / N = 50

	CHEST	ABDOMEN	ARM
CHEST	1.00000 0.0000	0.61986 0.0001	0.52973 0.0001
ABDOMEN	0.61986 0.0001	1.00000 0.0000	0.42925 0.0019
ARM	0.52973 0.0001	0.42925 0.0019	1.00000 0.0000

COVARIANCES AND CORRELATIONS 3
OUTPUT DATASET FROM PROC CORR

OBS	_TYPE_	_NAME_	CHEST	ABDOMEN	ARM
1	COV	CHEST	27.6295	23.4295	4.13622
2	COV	ABDOMEN	23.4295	51.7091	4.5852
3	COV	ARM	4.13622	4.5852	2.20663
4	MEAN		14.41	16.01	4.15
5	STD		5.25638	7.1909	1.48547
6	N		50	50	50
7	CORR	CHEST	1	0.619859	0.529728
8	CORR	ABDOMEN	0.619859	1	0.42925
9	CORR	ARM	0.529728	0.42925	1

REFERENCES

Blum, J.R., Kiefer, J., and Rosenblatt, M. (1961), "Distribution free tests of independence based on the sample distribution function," *Annals of Mathematical Statistics*, 32, 485-498.

Hoeffding, W. (1948), "A non-parametric test of independence," *Annals of Mathematical Statistics*, 19, 546-557.

Hollander, M. and Wolfe, D. (1973), *Nonparametric Statistical Methods*, New York: John Wiley & Sons.

Knight, W.E. (1966), "A Computer Method for Calculating Kendall's Tau with Ungrouped Data," *Journal of the American Statistical Association*, 61, 436-439.

Noether, G.E. (1967), *Elements of Nonparametric Statistics*, New York: John Wiley & Sons.

Chapter 33

The DATASETS Procedure

Operating systems: All

ABSTRACT

You can use the DATASETS procedure without full-screen (NOFS) processing or with full-screen processing if you have a full-screen terminal. PROC DATASETS can list, rename, and delete all types of members in a SAS data library. DATASETS can also alter variable names and related variable information, such as labels, informats, and formats in SAS data sets (library members of DATA type). With full-screen processing PROC DATASETS can also rename and delete items in the catalog (a library member of CAT type).

INTRODUCTION

See the "SAS Files" chapter for information about SAS data libraries, especially the types of files (or members) that can reside in a SAS data library.

Use the DATASETS procedure to manage a SAS data library. With PROC DATASETS you can list, rename, and delete all types of members in a SAS data library. For SAS data sets (members of DATA type) PROC DATASETS can change variable names and related variable information, such as informats, formats, or labels.

Note: you cannot use PROC DATASETS to process SAS data libraries in tape format.

From an interactive session, you can use the full-screen processing capability of PROC DATASETS if your terminal is full screen. With full-screen processing, you specify changes to library members with codes entered on menus displayed by DATASETS. Changes to the library members are made as soon as you type and enter appropriate codes on the screen. In addition to other procedure operations, with full-screen processing PROC DATASETS can delete and rename items in the catalog (members of CAT type). See **Full-Screen Processing** below.

In batch or interactive sessions with no full-screen (NOFS) processing in effect, you specify changes to library members in associated procedure statements, such as DELETE, EXCHANGE, and MODIFY statements. Changes are effective at the end of the PROC step if no errors are encountered in the step or if the FORCE option is in effect. Changes to variables in a SAS data set (named in the MODIFY statement) take effect when PROC DATASETS processes all FORMAT, INFORMAT, LABEL, and RENAME statements specified for that SAS data set.[1] See **Processing with NOFS in Effect** below.

SPECIFICATIONS

If you are using full-screen processing, you must specify the PROC statement only and exclude the NOFS and KILL options (described below). Details for processing with full screen are given in **Full-Screen Processing** below. Processing with NOFS in effect requires the use of associated procedure statements. These statements are described in **Processing with NOFS in Effect**.

PROC DATASETS Statement

PROC DATASETS [LIBRARY=*libref*] [*options*];

where

LIBRARY=*libref*
DDNAME=
: refers to the SAS data library to be processed. If LIBRARY= is not specified, the current default SAS library (usually the WORK library) is processed. DDNAME= is equivalent to LIBRARY=.

options
: specify whether you use full-screen processing, affect the amount of output produced, or limit procedure processing to specified member types in the library. You can specify the following options:

MEMTYPE=*membertype*
MTYPE=
MT=
: specifies one or more member types that are available for processing. Valid *membertype* values are[2]

  ALL all member types (default)

  CAT catalogs

DATA	SAS data sets
GCAT	graphics catalogs
IMLWK	IML work space files
MODEL	model files

In the PROC statement, the MEMTYPE= option limits the procedure processing to those types specified. For example, if the library being processed contains SAS data sets and graphics catalogs, you can limit processing to SAS data sets with the MEMTYPE= option, as follows:

```
PROC DATASETS LIBRARY=REF1 MEMTYPE=DATA;
```

To specify more than one type, enclose the list in parentheses. For example,

```
PROC DATASETS LIBRARY=REF1 MEMTYPE=(DATA GCAT);
```

If not specified, MEMTYPE=ALL is in effect. (For more information on types of members that can reside in a SAS data library, see the "SAS Files" chapter.)

NOFS

specifies that you will not use full-screen processing. You will invoke the procedure in a line-prompt or display manager session. NOFS (NO Full Screen) is automatically in effect if you invoke PROC DATASETS for batch processing or if you use a terminal that is not full screen. NOFS is also in effect if you enter other procedure statements associated with PROC DATASETS or the KILL option (described below). For example,

```
PROC DATASETS LIBRARY=REF1 NOFS;
   DELETE BOOKS;
```

and

```
PROC DATASETS LIBRARY=REF1;
   DELETE BOOKS;
```

will execute with NOFS in effect because both PROC steps include the DELETE statement.

KILL

deletes **all** members in the SAS library except those having PROTECT= passwords.[3]

You cannot invoke DATASETS for full-screen processing if you include the KILL option.

The following options affect DATASETS processing only when you are **not** using full-screen processing (that is, with NOFS in effect):

FORCE	updates the SAS library even if errors are encountered during execution of any procedure statements. Note: if the NOREPLACE system option has been specified, the FORCE option is ignored, and the library is not updated.
NOLIST	suppresses listing the members that are available for processing on the SAS log.
NOWARN	suppresses the warning message that is issued if a member specified in a DELETE or SAVE statement, or

listed as the last member in an AGE statement, is not in the SAS library currently being processed.

FULL-SCREEN PROCESSING

The full capability of DATASETS is available with full-screen processing. To invoke DATASETS for full-screen processing you must use a full-screen terminal and specify the PROC statement only, for example,

```
PROC DATASETS LIBRARY=libref options;
```

Full-screen is automatically in effect if you:

1. invoke PROC DATASETS for interactive execution from a full-screen terminal, and
2. enter only the PROC statement, and
3. do not use the NOFS or KILL option on the PROC statement.

With full-screen processing you enter codes on menus displayed by the procedure to control operations performed in the specified SAS library.

The Datasets Menu

When invoked for full-screen processing, DATASETS displays the Datasets Menu. For example, the following statement

```
PROC DATASETS LIBRARY=REF1;
```

invokes DATASETS for full-screen processing and displays the following Datasets Menu:

```
                    Datasets Menu

Command ===>

                     Library: REF1

      CMD      MEMBER      MEMTYPE  OPERAND1      OPERAND2

       _       member1     type     ________      ________
       _       member2     type     ________      ________
               .
               .
       _       membern     type     ________      ________
```

Screen 33.1 Datasets Menu

The LIBRARY, MEMBER, and MEMTYPE fields are initially filled with information obtained from the PROC statement and the specified library. The *LIBRARY* field contains REF1, the libref specified in the PROC statement. The *MEMBER* field lists in alphabetical order all members in the SAS data library referenced by REF1. *MEMTYPE* lists each member type. If the MEMTYPE= option is specified on the PROC statement, only members of the specified type(s) are listed. The fields containing underscores are input fields (described in **Input Fields** below).

The command-line commands *Command* ===> is the command line. The following commands can be entered on the command line. Many of these commands can also be issued with your terminal's function keys. Initial function key settings vary depending on the type of terminal you use. Enter the KEYS command (described below) for a list of current function key settings used to issue command-line commands.

Command-line commands

BACKWARD
BA
: scrolls information on the menu back toward the first line.

BOTTOM
BOT
: scrolls forward until the last line of information is displayed at the bottom of the menu.

CANCEL
CAN
: cancels processing and terminates the current menu.

CURSOR
CUR
: returns the cursor to the command line. This command is designed to be executed with a function key. Issue the KEYS command (described below) to learn which function key is set to issue the CURSOR command.

DUPLICATE
DU
: duplicates a selected value. This command is designed to be executed with a function key and is used in conjunction with the SELECT command (see below). To duplicate a selected value move the cursor to the appropriate field and press the DUPLICATE function key. Issue the KEYS command (described below) for a list of current function key settings.

END
: executes and terminates processing the current menu.

FORWARD
FOR
: scrolls information on the menu toward the last line.

HELP
H
: displays information about menus and codes used with DATASETS during full-screen processing.

KEYFIELD *field*
KEYF
: designates the field to be searched with LOCATE and NEXT commands (see below). This command is designed to be issued with

a function key. Enter the KEYS command for a list of current function key settings.

KEYS
K
: displays the current function key settings and allows you to change the settings and store them in a PROFILE member of the SASUSER library. See **Function Keys** and **SAS User Profile** sections of the "Full-Screen Editing Appendix" for information on using the function key definition screen and on storing your altered function key settings.

LOCATE *string*
L
: designates the character string that you want to find in the field specified by the KEYFIELD command.

NEXT
N
: finds the next occurrence of the string specified by the LOCATE command in the field designated by the KEYFIELD command.

RECALL
REC
: recalls up to 16 commands previously executed from the command line.

SELECT
SEL
: selects a value to be duplicated in another field when you press the DUPLICATE function key. The SELECT command is designed to be executed with a function key.

TOP
T
: scrolls to the first line of available library information.

X
: allows you to enter a terminal subsession from the full-screen menu in order to execute operating system commands. Enter the RETURN or END operating system command to return to DATASETS processing.

The message line The *message line* is located directly below the *Command line*. During processing, appropriate messages (errors or notes) appear on this line.

Input fields This menu contains three input fields: CMD, OPERAND1, and OPERAND2. *CMD* is the command field, actually a column of fields, that allows you to select the member to be processed. There is also an *OPERAND1* field and an *OPERAND2* field for each member. These fields are input fields used to specify information required for processing.

Selection codes You select a member by entering one of the following selection codes in the CMD field adjacent to the member name.

D
: to DELETE a member. Enter D in the CMD field adjacent to the member to be deleted. You can delete more than one member in a single operation. If the member is password protected, you must specify the write-password in the OPERAND1 field.[4]

  When the SAS data set is successfully deleted, the word "DELETED" will appear in the MEMTYPE field adjacent to the SAS

data set name. No further processing for the deleted SAS data set is allowed.

P

to set, reset, or remove a SAS data set PASSWORD.[5] Enter P in the CMD field adjacent to the appropriate data set. The MEMTYPE field must contain type DATA.

To set a password for a data set that is not password protected, enter the read-password in the OPERAND2 field and the write-password in the OPERAND1 field. You can specify either or both, but you must enter the write-password in the OPERAND1 field and the read-password in the OPERAND2 field.

To reset the read- or write-passwords, enter the existing read-password in the OPERAND2 field and the existing write-password in OPERAND1 field. If an existing password is incorrectly entered, you will receive an error message. When the correct password is entered, you will be prompted for the new password. When you are prompted, all other fields are protected; that is, you can enter the password(s) only. To remove password protection, enter blanks in OPERAND1 and OPERAND2 fields when you are prompted.

R

to RENAME a member. Enter R in the CMD field adjacent to the old member name and enter the new name in the OPERAND1 field. When the operation is complete, the new name replaces the old name in the MEMBER field.

You can rename more than one member in a single operation.

Use the following codes to select a member of a specific type and display a new screen containing information for that member:

B

to BROWSE a SAS data set (a member of DATA type). To select a SAS data set to be browsed, enter B in the CMD field adjacent to the member name. The MEMTYPE field must contain type DATA.

If the SAS data set is password protected, you must specify the read-password in the OPERAND2 field.[6]

When you select a SAS data set to browse, DATASETS displays a screen containing a list of SAS data set observations with values for each variable. All fields on this screen, except the command line, are protected; that is, you can enter only command-line commands when you browse a SAS data set. Enter the END command to return to the Datasets Menu.

C

to select a catalog (a member of CAT type). Enter C in the CMD field adjacent to the member name. The MEMTYPE field must contain type CAT.

When you select a CAT member, DATASETS displays a screen containing a list of items in the catalog. On this screen you can enter two codes:

R to RENAME catalog items

D to DELETE catalog items.

You enter R or D in the left column adjacent to the name of the item to be renamed or deleted.

The R and D codes used on the catalog screen process items in the selected catalog, while R and D codes entered in the Datasets

Menu process library members. For example, you can enter R on the catalog screen to rename an item in the catalog; you can enter R on the Datasets Menu to rename the catalog itself.

You can issue any of the command-line commands (described above) on this menu. Issue the END command to return to the Datasets Menu.

V

to select a SAS data set (a member of DATA type) in which VARIABLE information is to be updated. Enter a V in the CMD field adjacent to the SAS data set to be modified. The MEMTYPE field must contain type DATA. If the SAS data set is password protected, you must specify write- and read-passwords in OPERAND1 and OPERAND2 fields, respectively.[7]

When V is entered, PROC DATASETS displays the **Dataset Contents** menu:

```
                        Dataset Contents

Command ===>

  Library: libref                          Obs: n
   Member: membername                Created on: date
     Type: specialtype               Created by: userid or proc
    Label: label

              Type
#  Variable   Label     Length  Position   Format    Informat

1  name1      type        n        n       format    informat
              label
2  name2      type        n        n
              label
     .
     .
n  namen      type        n        n       format
              label
```

Screen 33.2 Dataset Contents Menu

As you can see, the Dataset Contents menu contains information similar to that which would be obtained if you ran PROC CONTENTS on the selected SAS data set. Menu fields are initially filled with current information about the selected SAS data set. You can enter or change variable information in the VARIABLE, FORMAT, INFORMAT, and LABEL fields only.

You can also change *specialtype* in the Type: field. (Valid *specialtypes* are DATA, CORR, COV, SSCP, EST, FACTOR, and DISCAL. See "SAS Files" for a discussion of special SAS data sets.)

To make changes simply type over the information shown in these fields.

Be aware that information entered or changed is not checked by the procedure. For example, DATASETS does not verify a format entered in the FORMAT field as a valid format name.

You can use any of the command-line commands on this menu. Enter the END command to return to the Datasets Menu.

Output

With full-screen processing PROC DATASETS issues messages and notes on the menu being processed as appropriate. No printed output is produced.

PROCESSING WITH NOFS IN EFFECT

When you use PROC DATASETS in batch or interactive (display manager or line-prompt) mode, you use options and associated statements to control DATASETS processing in the specified SAS data library. With NOFS in effect PROC DATASETS uses the following statements:

PROC DATASETS LIBRARY=*libref options;*
DELETE *members (options) ... / options;*
SAVE *members (options) ... / options;*
CHANGE *oldname=newname ... / options;*
EXCHANGE *name=anothername ... / options;*
AGE *currentname name2 ... lastname (options) / options;*
MODIFY *SASdataset (options);*
FORMAT *variable ... format. ... ;*
INFORMAT *variable ... informat. ...;*
LABEL *variable=newlabel ... ;*
RENAME *variable=newname ... ;*

The PROC DATASETS statement is described in **SPECIFICATIONS** above.

Many of the statements associated with PROC DATASETS allow options to be specified in parentheses or after a slash (/).

Options specified after the / provide default values for all members listed in the statement. Options specified in parentheses after a member name(s) override any options that follow the /. Also, it is important to note that options in parentheses apply to all preceding members in the list since the last options specification. For example, in the following statements

PROC DATASETS LIBRARY=*libref;*
DELETE member1 member2 (*option1*) member3 (*option2*);

option1 applies to member1 and member2, and *option2* applies to member3.

In associated statements you can specify the following options as appropriate:

PROTECT= specifies the write-password for members of DATA type (SAS data sets).[8]

MEMTYPE= selects one type of library member to be processed in this statement. If MEMTYPE= is specified in the PROC statement as well, you must use one of these previously specified types for MEMTYPE= in the associated statement.

If MEMTYPE= is not specified in the PROC statement, you can use any valid value for MEMTYPE= on the associated statement.

Using the MEMTYPE= option in an associated statement is slightly different from using MEMTYPE= in the PROC statement:

- You can specify only one type with the MEMTYPE= option in an associated statement.
- The default value for MEMTYPE= in associated statements is DATA if MEMTYPE= is not specified on the PROC statement or the associated statement.

Use of the MEMTYPE= option is discussed further with each of the following statements.

Use the following statements to control DATASETS processing when NOFS is in effect.

DELETE Statement

DELETE *members (options) ... /options;*

Use the DELETE statement to specify members to be deleted from the SAS library. For example, say you want to delete the members ONE and TWO from the SAS library referenced by the libref DIGIT:

```
PROC DATASETS LIBRARY=DIGIT;
   DELETE ONE TWO;
```

You can use an abbreviated member list where appropriate. For example:

```
DELETE RDU1-RDU5;
```

If you want to delete SAS data sets that are password protected, you must specify PROTECT=*password* in the DELETE statement or the PROC statement.[9]

Use MEMTYPE= in the DELETE statement to restrict processing to one type. If MEMTYPE= is specified in the PROC statement, use MEMTYPE= in the DELETE statement to further restrict processing to one of those previously specified types. For example, the MEMTYPE= option is used in both the PROC statement and the DELETE statement in the following program:

```
PROC DATASETS LIBRARY=FOOD MEMTYPES=(DATA GCAT);
   DELETE MEATS DAIRY / MEMTYPE=GCAT;
```

When this program executes, only MEATS and DAIRY of type GCAT will be deleted. If MEMTYPE= had not been specified on the DELETE statement, SAS data sets (DATA type) and GCAT members named MEATS and DAIRY would have been deleted.

If MEMTYPE= is not specified on the DELETE statement or the PROC statement, the default value is DATA; that is, only SAS data sets can be deleted.

If you attempt to delete members of a type that has not been specified on the PROC statement, you will receive an error message that the member cannot be found.

SAVE Statement

SAVE *members (option) ... / option;*

If you want to **delete** most of the members of a SAS data library, specify the names of the members you want to keep in the SAVE statement. SAS data sets that are password protected cannot be deleted using the SAVE statement, because you cannot specify the password. However protected SAS data sets can be saved without specifying their passwords.[10]

Because the SAVE statement provides a simple way to delete many members in one operation, be sure that you understand how the MEMTYPE= option affects which member types are saved and which types are deleted. The MEMTYPE= option in the SAVE statement specifies the type(s) of members that are deleted. The members named in the SAVE statement are saved while all other members of the specified type are deleted. For example, suppose you have a SAS data library containing members of type DATA, CAT, and GCAT. The following program

```
PROC DATASETS LIBRARY=FRUIT;
   SAVE APPLES ORANGES PEARS (MEMTYPE=CAT);
```

deletes all members of CAT type except APPLES, ORANGES, and PEARS. No members of any other type are deleted.

If you do not specify MEMTYPE= in the PROC statement or in the SAVE statement, MEMTYPE=DATA is in effect. For example,

```
PROC DATASETS LIBRARY=FRUIT;
   SAVE APPLES ORANGES PEARS;
```

deletes all members of DATA type in the library (referenced by FRUIT) except those named APPLES, ORANGES, and PEARS.

If you specify MEMTYPE= in the PROC statement, the MEMTYPE= value in the SAVE statement must be one of those previously specified types. For example,

```
PROC DATASETS LIBRARY=FRUIT MEMTYPE=(DATA CAT);
   SAVE APPLES ORANGES PEARS (MEMTYPE=CAT);
```

deletes all members of DATA type and all members of CAT type except APPLES, ORANGES, and PEARS. If MEMTYPE=CAT had not been specified in the SAVE statement in the last example, all members of CAT type except APPLES, ORANGES, and PEARS and all members of DATA type except APPLES, ORANGES, and PEARS would have been deleted.

Abbreviated member lists (for example, STUDY1-STUDY10) can be used with the SAVE statement.

CHANGE Statement

CHANGE *oldname=newname .../ options;*

Use the CHANGE statement to rename one or more members. Specify the old member name on the left side of the equal sign and the new member name on the right. For example, the statements

```
PROC DATASETS LIBRARY=OIL;
   CHANGE PERSIA=IRAN RUSSIA=USSR;
```

change the name of the SAS data set PERSIA to IRAN and RUSSIA to USSR.

Use the MEMTYPE= option to restrict name changes to one type of member. If MEMTYPE= is not specified in the CHANGE statement, types specified by MEMTYPE= in the PROC statement can be changed. If MEMTYPE= is not specified in the PROC statement or the CHANGE statement, the default type is DATA.

EXCHANGE Statement

EXCHANGE *name=anothername ... / options;*

Use the EXCHANGE statement to exchange the names of two members in the SAS library. For example, if your SAS library contains the members A and Z, the statements

```
PROC DATASETS LIBRARY=MYLIB;
   EXCHANGE A=Z;
```

would change the name of the member originally called A to Z, and Z to A.

More than one pair of member names can be changed with one EXCHANGE statement.

Use the MEMTYPE= option to restrict name exchanges to one type of member. If MEMTYPE= is not specified on the EXCHANGE statement or the PROC statement, DATA is the default type.

AGE Statement

AGE *currentname name2 name3 ... lastname (options)/options;*

If you have a group of related members in a data library to which members are periodically added, you can use the AGE statement to rename the members in the group. Then programs can access the latest addition to the group using one name that does not change.

When an AGE statement is used, the name of the first member is changed to the second name; the name of the second member is changed to the third name, and so on until the name of the next-to-last member is changed to the last name. The last member is then deleted.

For example, say that each day you run a SAS program that creates the SAS data set TODAY. You keep that SAS data set in a SAS library with seven other members. Each time you want to create a new SAS data set, the oldest SAS data set must be deleted and the remaining members renamed, leaving the name TODAY available for the new SAS data set. You can delete and rename the members using PROC DATASETS with the AGE statement:

```
PROC DATASETS LIBRARY=DAILY;
   AGE TODAY DAY1-DAY7;
```

After PROC DATASETS is executed, the SAS data set originally named TODAY is renamed DAY1; the SAS data set originally named DAY1 is renamed DAY2; and so on. The SAS data set originally named DAY6 is renamed DAY7, and the SAS data set originally named DAY7 is deleted. Since there is no longer a member named TODAY, the name TODAY is available for the new SAS data set.

Notice that the above example uses an abbreviated SAS data set list, DAY1-DAY7.

If you are aging SAS data sets that are password protected, you must specify the PROTECT= option in the AGE statement for the one to be deleted (that is, the last one in the list).[11]

Use the MEMTYPE= option to specify one type of member to age.

If the first member named in the AGE statement is not in the SAS library, the AGE statement does not execute. If other members listed are not present, the SAS System prints a note on the log and renames all the members that are present.

MODIFY Statement

MODIFY SAS*dataset (options);*

where *SASdataset* is a member of DATA type.

Use the MODIFY statement and options when you want to change attributes in the specified SAS data set. You can specify only one SAS data set name in each MODIFY statement, and you cannot specify the same data set name in more than one MODIFY statement in a given PROC step.

The MODIFY statement is used in conjunction with other procedure statements (described below) to change variable names, labels, informats, or formats in the specified SAS data set. Descriptor records are updated in place, so no extra space is needed for the SAS data set. History records are not updated, and the number of generations does not increase.

The following options can be specified in the MODIFY statement. Options must be enclosed in parentheses.

(LABEL=*newlabel)*
(LABEL=)
: specifies a *newlabel* or removes a label for the SAS data set named in the MODIFY statement. A new label cannot be longer than 40 characters and must be enclosed in single quotes.[12]

  If you enter LABEL= with no value, the old label is removed but not replaced. For example,

```
PROC DATASETS LIBRARY=TEST;
   MODIFY EXP1 (LABEL=' ');
   MODIFY EXP2 (LABEL='TESTS ON THREE YEAR OLDS');
```

  The label of the SAS data set TEST.EXP1 is removed by the first MODIFY statement and the label of TEST.EXP2 is changed in the second MODIFY statement.

(PROTECT=*writepassword)*
(PROTECT=*old/new)*
: specifies, changes, or removes the *writepassword* for the SAS data set named in the MODIFY statement.[13]

  You must specify *(PROTECT=writepassword)* to access a SAS data set that is write-password protected. To change an existing write-password, specify *(PROTECT=old/new)*. To remove a password, just enter the old password and a slash. For example,

```
PROC DATASETS LIBRARY=TEST;
   MODIFY EXP1 (PROTECT=RED/);
   MODIFY EXP2 (PROTECT=GREEN/BLUE);
```

  The write-password for TEST.EXP1 is removed by the first MODIFY statement and the password for TEST.EXP2 is changed to BLUE by the second MODIFY statement.

  PROTECT= can also be used in the MODIFY statement to specify a password for a previously unprotected SAS data set. Simply specify the password as you would an existing password. For example, if EXP7 is unprotected, you can specify

```
MODIFY EXP7 (PROTECT=SAFE);
```

  to assign the write-password SAFE to the SAS data set.

(READ=*readpassword)*
(READ= *old/new)*
: specifies, changes, or removes the *readpassword* for the SAS data set named in the MODIFY statement.[14]

  If a read-password exists for a SAS data set, it must be specified with the READ= option or PROC DATASETS will not access the SAS data set. To change a read password, specify the *old* password after READ=, followed by a slash and a *new* password. To remove a password, give the old password and a slash. The password is removed but not replaced. For example,

```
PROC DATASETS LIBRARY=TEST;
   MODIFY EXP1 (READ=NCS/DU);
   MODIFY EXP2 (READ=APPLE/);
```

The first MODIFY statement changes the read-password for TEST.EXP1 to DU, and the second MODIFY statement removes the read-password for TEST.EXP2.

READ= can also be used in the MODIFY statement to assign a password to a previously unprotected SAS data set. Simply specify the password as you would an existing password. For example, if EXP7 has no read protection, you can assign a password with this MODIFY statement:

```
MODIFY EXP7 (READ=SECURE);
```

(TYPE=*specialtype*)
assigns or changes the type of special SAS data set created by some SAS procedures (for example, PROC CORR and PROC FACTOR) or in a DATA step.

Valid *specialtypes* are DATA, CORR, COV, SSCP, EST, FACTOR, and DISCAL. See "SAS Files" for a discussion of special SAS data sets.

Note: do not confuse TYPE= with MEMTYPE=. TYPE= specifies a type of special SAS data set. MEMTYPE= specifies one or more types of members in a SAS data library.

You must be sure that the specified value of TYPE= corresponds to the actual SAS data set type. The SAS data set type specified by TYPE= is not validated by SAS (except to check if it is a length of eight characters or less). SAS does not verify that the SAS data set's structure is appropriate for the type you have designated, or even that you have specified a valid special type.

Use the following statements to change variable labels, names, informats, and formats in a SAS data set named in the MODIFY statement.

FORMAT Statement

FORMAT *variable ... format. variable ... format. ...;*

Variable formats can be changed or removed with the FORMAT statement. To change a format, specify the name of the variable and then the new format. When the variable name is given but there is no accompanying format, the old format is removed, but not replaced. Multiple variables can be used, and abbreviated variable lists, such as X1-X5, can be used where appropriate. You can change as many formats as you want with one FORMAT statement. Here is an example:

```
PROC DATASETS LIBRARY=STUDY;
   MODIFY GROUP1 (PROTECT=CCCDDD);
   FORMAT X1-X3 4.1 TIME HHMM2.2 AGE;
```

The SAS data set GROUP1 is a member of the SAS library STUDY. The password CCCDDD must be given to access the data set. Variables X1, X2, and X3 are assigned the 4.1 format; the variable TIME is assigned the HHMM*w.d* format; and the format of the AGE variable is removed.

Note that you must list any variables whose formats are to be removed last in the FORMAT statement; otherwise, they are assigned the first format that follows. For example, if you write the FORMAT statement from the example above as

```
FORMAT X1-X3 4.1 AGE TIME HHMM2.2;
```

the AGE variable would have its format changed to HHMM2.2, rather than removed.

INFORMAT Statement

INFORMAT *variable ... informat. variable ... informat. ...;*

Informats can be changed or removed using the INFORMAT statement. To change an informat, specify the name of the variable and then the new informat. When a variable name is specified but no accompanying informat is specified, the old informat is removed.

Multiple variables and abbreviated variable lists can be used. For example, in the SAS data set GROUP1 the variables A, B, and X1-X3 are assigned new informats, and the variable C has its informat removed, with these statements:

```
PROC DATASETS LIBRARY=STUDY;
   MODIFY GROUP1 (PROTECT=CCCDDD);
   INFORMAT A B 2. X1-X3 4.1 C;
```

Note that you must list any variables whose informats are to be removed last in the INFORMAT statement; otherwise they are assigned the first informat that follows. For example, if you write the above INFORMAT statement as

```
INFORMAT A B C 2. X1-X3 4.1;
```

the variable C has its informat changed to 2., rather than removed.

LABEL Statement

LABEL *variable=newlabel ...;*

The LABEL statement changes or removes variable labels. You can use as many variables as you want in the LABEL statement, as long as each variable name is followed by an equal sign. To specify a new label, follow the equal sign with a new label. To remove a label do not specify anything after the equal sign. If the new label includes a right parenthesis, equal sign, or semicolon, it must be enclosed in single quotes. If a single quote appears in the label, it must be written as two single quotes in the LABEL statement. The new label can be up to 40 characters long.

The statements

```
PROC DATASETS LIBRARY=STUDY;
   MODIFY GROUP1 (PROTECT=CCCDDD);
   LABEL X1='Score 1=' X2='Score 2=' A=;
```

change the labels for variables X1 and X2 and remove the label for variable A.

RENAME Statement

RENAME *variable=newname ...;*

Variables can be assigned new names using the RENAME statement. The new name must be a valid SAS name. There is no limit to the number of variables that you can rename in one statement. For example,

```
RENAME DAY1=TIME1 DAY2=TIME2;
```

Output

When NOFS is in effect, the DATASETS procedure lists the members in the SAS library before and after the library is updated unless the NOLIST option is speci-

fied. The list appears on the SAS log. If the MEMTYPE= option is used, only specified types are listed.

EXAMPLES

Using PROC DATASETS with NOFS in Effect: Example 1

There are several ways to execute PROC DATASETS with NOFS in effect. You can include the appropriate control language and submit the following job for batch execution. You can create a file containing the program and specify the file's name when you invoke SAS for noninteractive execution. You can invoke the SAS System to execute the following program in line-prompt mode. Or, if you have a full-screen terminal, you can invoke SAS in display manager mode, then enter and execute this program:

```
PROC DATASETS LIBRARY=MISC;
   DELETE GEORGIA;
   CHANGE GDPSTATS=ECONOMIC;
   MODIFY CARS(LABEL=CARS RATED ON SIX SCALES);
      RENAME NOISE=QUIET;
      LABEL ACCEL=ACCELERATION
         VISIBLE=VISIBILITY
         CARGO=CARGO SPACE;
   MODIFY COFFEE(TYPE=CORR);
      FORMAT DEEP BREWED HEARTY PURE ROASTED FRESH 4.2;
```

This program processes a SAS library referenced by libref MISC. Since the MEMTYPE= option is not specified on the PROC statement or on associated statements, only DATA types (SAS data sets) are available for processing. First, DATASETS produces a list of SAS data sets on the SAS log. Then the procedure deletes a SAS data set named GEORGIA, changes a SAS data set name from GDPSTATS to ECONOMIC, and modifies characteristics of two SAS data sets. DATASETS then writes the updated SAS data set list on the SAS log.

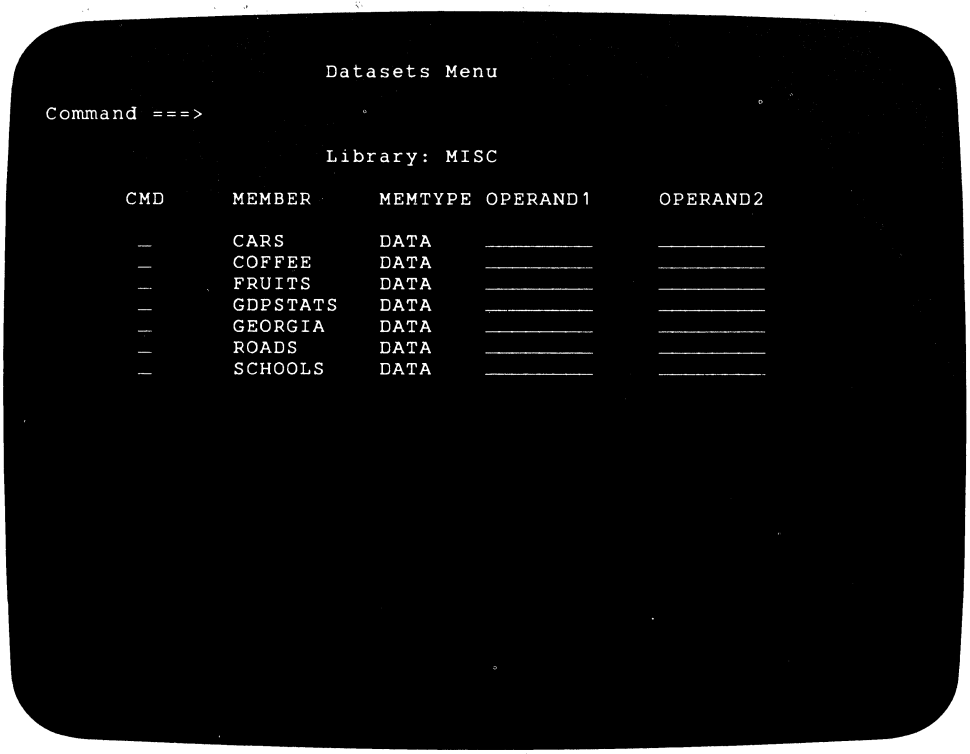
Using PROC DATASETS in Full-Screen Mode: Example 2

Suppose you have a full-screen terminal and decide to execute the program given in Example 1 using PROC DATASETS with full-screen processing. First, invoke the SAS System and enter the following statement:

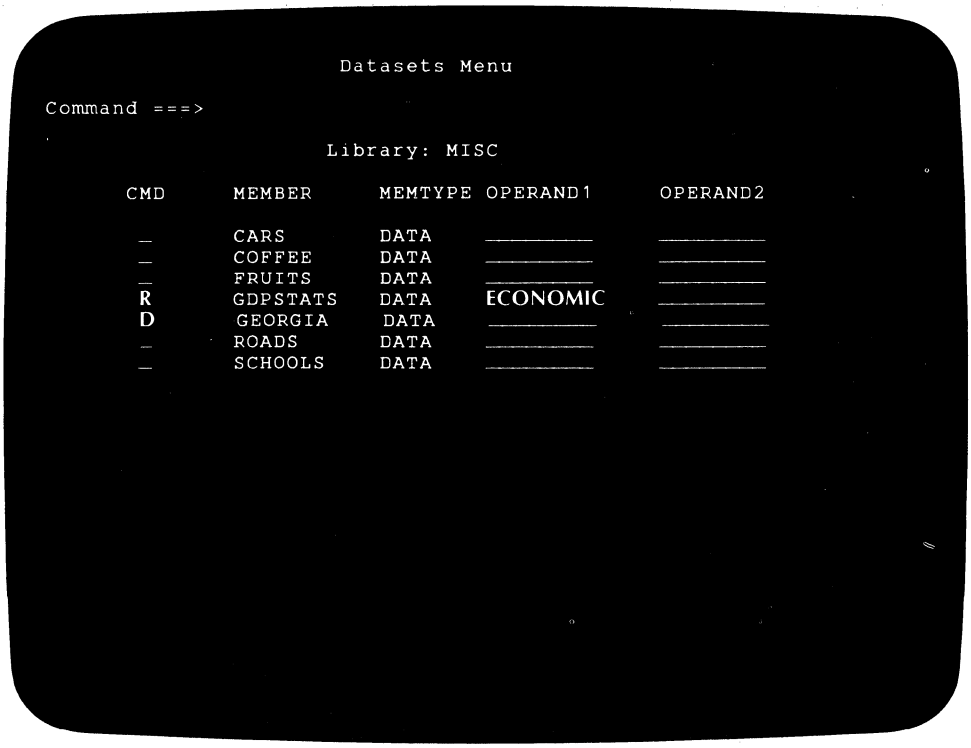
```
PROC DATASETS LIBRARY=MISC MEMTYPE=DATA;
```

Notice that PROC DATASETS is limited to processing DATA types by including the MEMTYPE=DATA specification. **Screen 33.3** is displayed with a list of all DATA type members contained in the SAS library referenced by MISC.

To accomplish the tasks performed in **Example 1** you must delete the SAS data set GEORGIA, change a SAS data set name from GDPSTATS to ECONOMIC, AND MODIFY SAS data sets CARS and COFFEE. The first two operations are indicated in boldface on **Screen 33.4**.



Screen 33.3 Datasets Menu before Processing



Screen 33.4 Datasets Menu Showing Rename and Delete Features

The D entered in the CMD field deletes the SAS data set named GEORGIA. The SAS data set GDPSTATS is renamed by entering R in the CMD field adjacent to GDPSTATS and typing the new name ECONOMIC in the OPERAND1 field. When this menu is executed, ECONOMIC replaces GDPSTATS in the MEMBER field, and "DELETED" appears in the MEMTYPE field adjacent to the SAS data set GEORGIA. The CMD field clears.

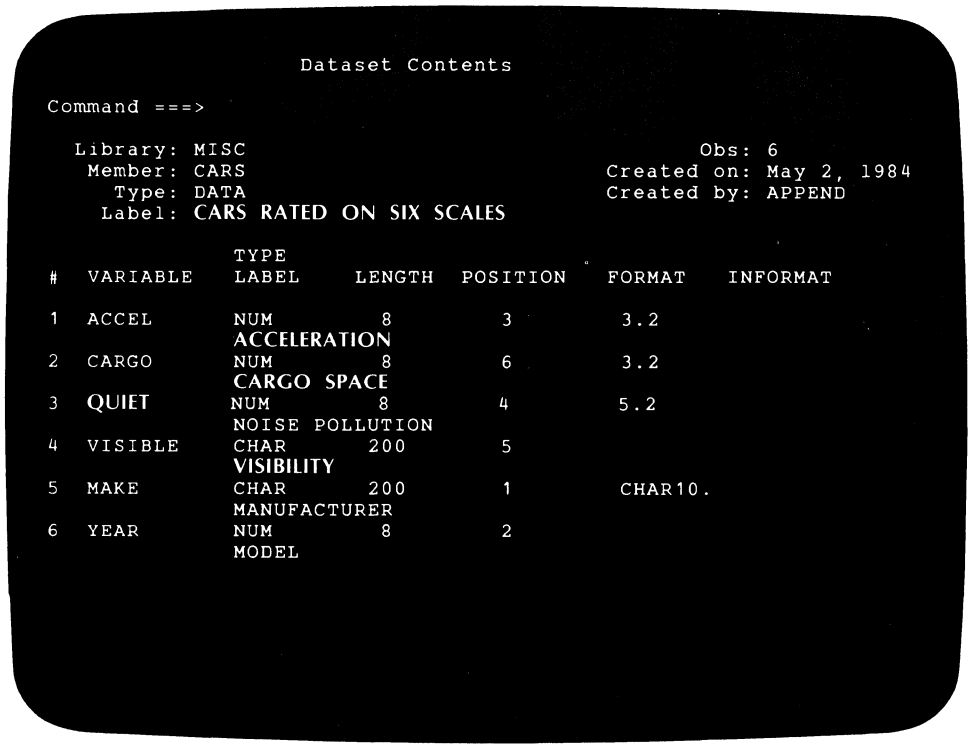
To modify the SAS data set CARS, enter a V in the CMD field adjacent to the member name. The following Dataset Contents menu is displayed for CARS.

```
                        Dataset Contents

Command ===>

  Library: MISC                                        Obs: 6
   Member: CARS                                Created on: May 2, 1984
     Type: DATA                                Created by: APPEND
    Label: CARS RATED

                Type
#  Variable     Label      Length   Position    Format    Informat

1  ACCEL        NUM             8          3     3.2
                SPEED
2  CARGO        NUM             8          6     3.2
                SPACE
3  NOISE        NUM             8          4     5.2
                NOISE POLLUTION
4  VISIBLE      CHAR          200          5
                ABILITY TO SEE
5  MAKE         CHAR          200          1     CHAR10.
                MANUFACTURER
6  YEAR         NUM             8          2
                MODEL
```

Screen 33.5 Dataset Contents before Processing CARS Data Set

To change information on the screen simply move the cursor to the appropriate field and type over the information that is shown. **Screen 33.6** shows the changes in boldface. When all changes for CARS are made, issue the END command to return to the Datasets Menu.

To select the data set COFFEE enter V in the CMD field of the Datasets Menu. PROC DATASETS will then display a Dataset Contents menu for SAS data set COFFEE. You make changes to COFFEE (as indicated in Example 1) by typing 4.2 in the FORMAT field for the appropriate variables. When you have completed processing for COFFEE, enter END to return to the Datasets Menu. This completes the work for Example 1. You can now select other members for processing or issue END from the Datasets Menu to terminate full-screen processing.



```
                    Dataset Contents

Command ===>

  Library: MISC                                        Obs: 6
   Member: CARS                                  Created on: May 2, 1984
     Type: DATA                                  Created by: APPEND
    Label: CARS RATED ON SIX SCALES

                TYPE
#  VARIABLE     LABEL     LENGTH   POSITION   FORMAT    INFORMAT

1  ACCEL        NUM            8          3   3.2
                ACCELERATION
2  CARGO        NUM            8          6   3.2
                CARGO SPACE
3  QUIET        NUM            8          4   5.2
                NOISE POLLUTION
4  VISIBLE      CHAR         200          5
                VISIBILITY
5  MAKE         CHAR         200          1   CHAR10.
                MANUFACTURER
6  YEAR         NUM            8          2
                MODEL
```

Screen 33.6 Dataset Contents after Processing CARS Data Set

NOTES

1. **OS and VSE**: When PROC DATASETS changes a SAS data library, the library's directory is updated to reflect the changes.

2. **AOS/VS, PRIMOS, and VMS**: In addition to those listed you can also specify

FORMATC	character formats
FORMATN	numeric formats

3. **AOS/VS, PRIMOS, and VMS**: Password protection for SAS library members is not implemented.

4. See note 2.

5. **AOS/VS, PRIMOS, and VMS**: The P selection code is ignored because password protection for SAS library members is not implemented.

6. See note 2.

7. See note 2.

8. See note 2.

9. See note 2.

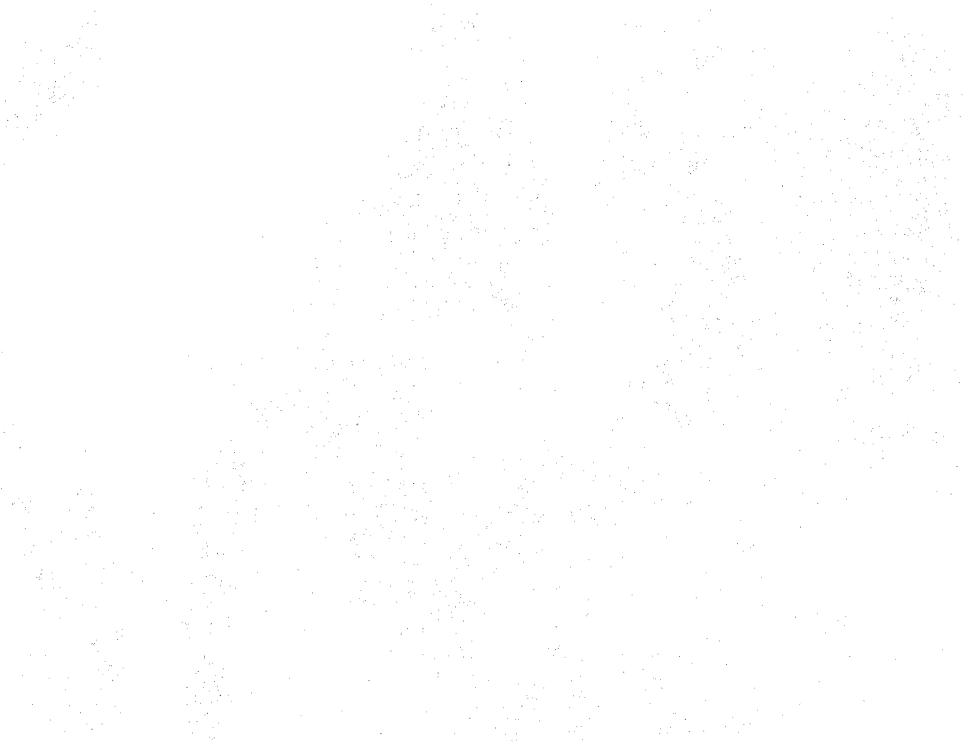
10. See note 2.

11. See note 2.

12. **CMS, OS, and VSE**: The label must be enclosed in quotes only if it contains parentheses, semicolons, or equal signs.

13. See note 2.

14. See note 2.



Chapter 34

The EDITOR Procedure

Operating systems: All

ABSTRACT

The EDITOR procedure is used to examine and make changes to a SAS data set without using a SAS DATA step. PROC EDITOR is primarily an interactive procedure.

INTRODUCTION

When a data set is edited in a DATA step, you must create a copy to make changes. The EDITOR procedure allows you to edit a SAS data set directly with search, list, addition, deletion, and replacement commands. Since PROC EDITOR is interactive, you get a response to each command as it is executed. (It is also possible to execute PROC EDITOR as a batch job.)

Use of PROC EDITOR is restricted to disk-format SAS data sets. Keep in mind that PROC EDITOR operates directly on the input data set. This means that as commands are executed, changes are made to the data set, and it is not possible to start again with the original data set.[1] If you make mistakes when changing a

data set, you cannot return to the original. Begin by making a backup copy of any data set that would be difficult to recreate if errors are made.

EDITOR vs. BROWSE

The EDITOR procedure and the BROWSE procedure are very similar, but they differ in two ways:

- The BROWSE procedure can be invoked on a data set using read access, but the EDITOR procedure must have write access to the SAS data set.
- No change commands (REPLACE, ADD, DELETE, DUP) are allowed with the BROWSE procedure; that is, you cannot edit a SAS data set with PROC BROWSE.

Introductory Example

In the following example, PROC EDITOR is used to edit the SAS data set SAVE.HTWT. Commands are entered in lowercase in response to line number prompts from the SAS System.[2] Responses from SAS are in uppercase. First, invoke the EDITOR procedure with the PROC statement.

```
1? proc editor data=save.htwt;
```

SAS responds:

```
WELCOME TO THE SAS LINE EDITOR. BEGIN ENTERING COMMANDS.
```

Ask SAS to list some observations using the LIST command (the responses are shown after each command):

```
2> list 1;
1 NAME=ALFRED SEX=M AGE=14 HEIGHT=69 WEIGHT=112.5;

3> list 1000;
NOTE: STARTING OBSERVATION EXCEEDS OBSERVATION COUNT.
USING LASTOBS.
NOTE: LAST OBSERVATION VALUE EXCEEDS OBSERVATION COUNT.
USING LASTOBS.
19 NAME=WILLIAM SEX=M AGE=15 WEIGHT=66.5 WEIGHT=112;
```

Since there are only 19 observations in the data set, SAS cannot list the one-thousandth observation, so it lists the last one.

Next, search for suspiciously high WEIGHT values using the FIND command:

```
5> find all 1,last weight>140;
```

SAS responds with observations having WEIGHT values greater than 140:

```
FOUND AT OBS 10 .
FOUND AT OBS 15 .
END OF SEARCH. OBS= 19 .
```

List the NAME and WEIGHT values for observation 10, then replace the WEIGHT value with 99.5, and list observation 10 again:

```
6> list 10 name weight;
10 NAME=JOHN WEIGHT=995;

7> rep weight=99.5;list;
10 NAME=JOHN SEX=M AGE=12 HEIGHT=59 WEIGHT=99.5;
```

Then list NAME and WEIGHT values for observation 15:

```
8> list 15 name weight;
15 NAME=PHILIP WEIGHT=150;
```

End the session with the END command:

```
9> end;
EXIT FROM SAS EDITOR.

10?
```

SPECIFICATIONS

The EDITOR procedure operates in two stages. The first stage involves the PROC, FORMAT, and INFORMAT statements. The PROC statement and associated FORMAT and INFORMAT statements are examined as a unit and then executed. PROC EDITOR is then in interactive mode.

The interactive mode is the second stage of the procedure and involves commands that are executed immediately after they are entered.

Below are the statements and commands used with PROC EDITOR.

PROC EDITOR *options;*
FORMAT *variable format. ...;*
INFORMAT *variable informat. ...;*
FIND *options range variable1 operator1 value1...;*
LOCATE *options range value...;*
SEARCH *options range string...;*
NAME *variable;*
STRING *variable...;*
VERIFY *option;*
END;
LIST *range variable...;*
REPLACE *options range*
variable1=value1 variable2=value2...;
ADD *variable1=value1 variable2=value2...;*
DELETE *range;*
DUP *range;*
TOP;
BOTTOM;
UP *n;*
DOWN *n;*

You should not use the RUN statement in PROC EDITOR under Version 5. SAS ignores RUN statements entered before the first EDITOR command. After an EDITOR command, using RUN produces the same result as using the END or QUIT statements: all of these end your editing session with PROC EDITOR. Use only the END or QUIT statements for that purpose.

PROC EDITOR Statement

PROC EDITOR *options;*

The following options can be specified on the EDITOR statement.

DATA=*SASdataset*
names the SAS data set to be edited. If DATA= is omitted, the most recently created data set is used.

FUZZ=*value*
specifies the "fuzz" value to be used in the FIND command. See the FIND command for further explanation.

FORMAT Statement

FORMAT *variable format. ...;*

Use the FORMAT statement to assign output formats to variables for the duration of your PROC EDITOR session. (See "Statements Used in the PROC Step" for a complete discussion of the FORMAT statement.) The maximum widths allowed by PROC EDITOR are 16 for numeric variables and 64 for character variables. The width restrictions on output formats are important for the PROC EDITOR commands that display values; that is, the REPLACE command with the VERIFY command in effect and the LIST command. (REPLACE and LIST are discussed later in this chapter.)

If formats were assigned to variables when the data set was created, it is not necessary to repeat the FORMAT statement for the EDITOR procedure.

Date, time, and datetime variables must be assigned a corresponding date, time, or datetime format if you want them to be printed in a readable form.

INFORMAT Statement

INFORMAT *variable informat. ...;*

The INFORMAT statement assigns informats to variables for the duration of your PROC EDITOR session. (See "Statements Used in the DATA Step" for a complete discussion of the INFORMAT statement.) If informats were assigned to variables when the data set was created, it is not necessary to repeat the INFORMAT statement for the EDITOR procedure.

The maximum widths allowed for informats are 32 for numeric variables and 200 for character variables (the standard limits for informats). It is important to note, however, that PROC EDITOR uses up to 64 characters of an informatted character value only. If an informatted character value is 70 characters long, any searching or changing command in PROC EDITOR uses only the first 64 characters. The commands affected by this PROC EDITOR width restriction are FIND, LOCATE, SEARCH, REPLACE, and ADD.

If you do not specify an informat for a variable, PROC EDITOR applies the default value. If you do specify the value of a variable, PROC EDITOR uses the appropriate informat routine, default or otherwise, when:

1. the value is enclosed in quotes. In the following example, the FIND command will search for X='AB':

```
INFORMAT X $HEX.;
FIND 1,LAST X='C1C2';
```

   In the examples below, Y is a numeric variable. The two commands shown below are equivalent; they produce the same results:

```
FIND 1,LAST Y='1';

FIND 1,LAST Y=1;
```

   Note that giving the number without quotes is more efficient, however, since the quoted string must be processed through an informat routine.
2. a character informat is given and the character string is not enclosed in quotes, but is a valid SAS name. In the following example, if X has an informat of $HEX, the FIND command will search for values of X='AB'.

```
FIND 1,LAST X=C1C2;
```

If the informatted value is not a valid SAS name, it must be enclosed in quotes, regardless of whether the value is for a numeric or character variable. Consider the following examples:

```
INFORMAT Y DATE.;
FIND 1,LAST Y=01JAN85;

FIND 1,LAST Y='01JAN85';

FIND 1,LAST Y='01XXX85';

FIND 1,LAST Y='01JAN85'D;
```

The first FIND command is invalid, since 01JAN85 is not a valid SAS name. The second command is valid because the value is enclosed in quotes, and it would be properly recognized by the informat routine DATE. The third command is not valid, even though the value is enclosed in quotes, because 01XXX85 is not a valid DATE7. value. The last FIND command is valid, but '01JAN85'D would be treated as a number because date literals are translated to numbers before PROC EDITOR receives them. The last command, therefore, would be equivalent to

```
FIND 1,LAST X=9132;
```

Note that unless you enclose a string in quotes and use the NOCAPS system option, PROC EDITOR will convert the values you enter to uppercase automatically. Therefore, you should consider whether you want those values to be saved in upper- or lowercase.

For example, if a data set has the value *abc* in observation 5, and you enter

```
FIND 1,LAST X=ABC;
```

the value will not be located at observation 5. Neither will it find the value if you enter

```
FIND 1,LAST X='ABC';
```

If, however, you set NOCAPS ON and enter

```
FIND 1,LAST X='abc';
```

the match will be found. Note that this method must also be used with the ADD, LOCATE, REPLACE, and SEARCH commands.

The INFORMAT statement can be reissued at any time during the EDITOR session.

PROC EDITOR Commands

There are five types of EDITOR commands: searching commands, specification commands, display commands, change commands, and positioning commands. The commands in each category are

Searching: FIND, LOCATE, SEARCH
Specification: NAME, STRING, VERIFY, END
Display: LIST
Change: REPLACE, ADD, DELETE, DUP
Positioning: TOP, BOTTOM, UP, DOWN

There are options and other specifications with most of the EDITOR commands.

With many EDITOR commands it is possible to specify a **range** of observations to search, display, or change. The range specification follows the command keyword and is usually given by two numbers separated by a comma. If only one number is given, only that observation is searched. If no number is given, only the current observation is searched. (An exception to this rule is explained in **Repeating FIND, LOCATE, or SEARCH Commands**, below.) The keyword LAST can be used instead of the second number to indicate the last observation number. Here are some examples of ranges:

1,5	from observation 1 through observation 5
3,LAST	from observation 3 through the last observation
6	observation 6 only

Each EDITOR command is explained below, under the appropriate category.

Searching commands The searching commands are those that search the SAS data set for an occurrence of a set of values specified in the command. The general form of a searching command is

COMMAND *options range values;*

where *options* can be VERIFY or ALL (or both), *range* specifies the range of observations to search, and *values* is the list of values for which to search. The options are discussed below. The value list is explained with each command since it operates differently with each command.

The following options can be used with searching commands:

VERIFY
VER
: causes the NOGO switch to be turned on if a search is not successful. The switch prevents any further changes to the SAS data set (using REPLACE, ADD, DELETE, or DUP) until a VERIFY RESET; command is given. This feature is especially useful in a batch job since you cannot respond to EDITOR when it tells you that the values cannot be found.

ALL
: specifies a search for all matches within the given range. If ALL is not given, searching stops after the first match is found, and that observation becomes the current observation. When ALL is used, all observations in the designated range are searched, and the last observation becomes the current observation.

FIND command

FIND *options range*
variable1 operator1 value1 variable2 operator2 value2 ...;

where *operator* can be $=$, $\neg=$, $>$, $\geq$, $<$, or $\leq$.

The FIND command searches a given range of observations for occurrences of specified variable/value relationships. For example,

```
FIND 1,5 X=2 Y<6 Z¬=7;
```

searches observations 1 through 5 for the first observation in which X is equal to 2, Y is less than 6, and Z is not equal to 7.

The same variable can appear more than once in a value list. The FIND command

```
FIND ALL 1,LAST X≥5 X≤10;
```

searches for all observations in the data set that have X values between 5 and 10.

If none of the observations searched meets the specified conditions, the message

```
NOT FOUND. OBS=n .
```

is issued, where *n* is the number of observations in the data set.

Some searches involve comparisons between values that are very close—but not exact matches—because of the rounding that occurs when creating the data set. The FUZZ= option can be specified when entering PROC EDITOR to indicate how closely the searching value must match the variable value in order for it to be considered matched. The default fuzz value is 1E-12. Fuzzing is performed only if the = operator is used, and only on numeric variables. The type of fuzzing used is called "relative fuzzing." The conditional formula is

$$|x - y|/(|x| + |y|) < fuzz$$

If this condition is met, a match is found. For example, suppose the value of a variable Z in an observation is equal to 1234567890123, and FUZZ=1E−12 (the default). The command

```
FIND ID=1234567890124;
```

finds a match, since

```
|1234567890123-1234567890124|/(1234567890123+1234567890124)
```

is less than 1E−12.

For very large numbers, use larger FUZZ values. In the example above, if FUZZ=1E−15, the two values do not match.

See **Repeating FIND, LOCATE, or SEARCH Commands**, for an explanation of repeating a FIND command.

LOCATE command

LOCATE *options range value;*
LOC *options range value;*

The LOCATE command searches for the occurrence of a single value in a range of observations. A LOCATE command can be repeated at any time during an EDITOR session.

Before a LOCATE command can be issued, a NAME command must be given:

NAME *variable;*

The NAME command, described below under **Specification Commands,** gives the name of the variable that the LOCATE command evaluates. For example,

```
NAME X;
LOCATE 1,10 2.35;
```

searches the variable X in observations 1 through 10 for an occurrence of the value 2.35.

If a character value specified in the LOCATE command contains special characters (for example, blanks, underscores, or semicolons), the value string must be enclosed in single quotes.

If you specify a value in a LOCATE command, you must also specify a range. This is necessary because, for example, the statement

```
LOCATE 1;
```

is ambiguous. It could mean that you want SAS to locate a number 1 in the current observation, but it could also mean that you want SAS to locate a value—given in a previous LOCATE statement—in observation 1.

Below are some examples of LOCATE commands:

```
LOCATE 1,LAST ABC;
LOCATE 6,20 'X Y Z';
LOCATE 15,LAST 10;
```

The value given in the last example, 10, could be used for either a character or a numeric variable. If character, the procedure searches for the characters 10; if numeric, the procedure searches for the number 10.

The LOCATE command can also be used to search for leading characters only. Indicate a partial search by adding a P after the LOCATE keyword. For example, in a SAS data set containing data on individuals, there are names of the form "PUBLIC, JOHN Q." You are looking for anyone named SMITH, but you do not care about the first name. Enter

```
LOCATE P 1,LAST SMITH;
```

to look at the first five characters of each NAME value to see if they are SMITH. Note that EDITOR would also find matches with names such as SMITHFIELD. To access only those whose last name is exactly SMITH, enter

```
LOCATE P 1,LAST 'SMITH,';
```

so that names such as SMITHFIELD are not considered matches.

See **Repeating FIND, LOCATE, or SEARCH Commands** for a discussion of repeating the LOCATE command.

SEARCH command

SEARCH *options range string1 string2 ...;*
S *options range string1 string2 ...;*

The SEARCH command searches for occurrences of *strings* within a group of character variables. Before a SEARCH command is invoked, a STRING command must be given.

STRING *charactervariable ...;*

The STRING command, described below under **Specification Commands**, names the variables used by the SEARCH command. If you enter

```
STRING NAME SPOUSE;
SEARCH 1,LAST SMITH 'SMITH-JONES';
```

the variables NAME and SPOUSE are searched on all observations for the strings SMITH and SMITH-JONES.

A string can be found anywhere within a value, not just at the beginning.

The string list for a SEARCH command consists of 1 to 10 strings and cannot exceed 200 characters in total length. Only character variables can be searched by the SEARCH command. Single quotes are not necessary unless special characters appear in the string, or unless you are searching for lowercase characters.

Any string in the list can apply to more than one of the specified variables, so the string is processed by each variable's informat before a comparison is made.

There are two types of searching: "any" and "all." "Any" is indicated by a "@" after the word SEARCH. "All" is the default. In an "any" search, only one of the strings must be found in the variables specified in the STRING command. In an "all" search, all strings specified in the list must be found somewhere in the variables specified by the STRING command. Suppose, for example, that the variable NAME has the value PUBLIC, JOHN Q. and SPOUSE has the value PUBLIC, MARY J. Consider these commands:

```
STRING NAME SPOUSE;
SEARCH 'JOHN' 'MARY';
```

For this SEARCH command the string JOHN has to be found somewhere in either NAME or SPOUSE. It is found in NAME. In addition, MARY must be found in NAME or SPOUSE. It is found in SPOUSE. Therefore, the SEARCH is successful.

```
STRING NAME SPOUSE;
SEARCH 'JOHN' 'JANE';
```

In the example above, JOHN is found but JANE is not. Since SEARCH "all" is requested (by default) and no match is found, the SEARCH is not successful.

```
STRING NAME SPOUSE;
SEARCH@ 'JOHN' 'JANE';
```

For the third example, SEARCH "any" is requested. JOHN is found, so the search is successful.

```
STRING NAME SPOUSE;
SEARCH@ 'JANE';
```

In the final example, JANE is not found. Since no strings were found, the SEARCH "any" request is not successful.

See **Repeating FIND, LOCATE, or SEARCH Commands** for a description of repeating the SEARCH command.

Specification commands The specification commands are those that do not (for the most part) display information and do not operate on the SAS data set. These commands are NAME, STRING, VERIFY, and END.

NAME command

NAME *variable;*

The NAME command precedes the LOCATE command. It names a single variable that LOCATE should search for and match. The specified variable can be numeric or character. If no variable name is given, the command displays the name of the current NAME variable. Only one variable name can be specified.

The NAME command can be reissued as often as necessary. It does not have to be reissued between uses of the LOCATE command or any other searching command. For example, if you are looking for occurrences of two particular values of the variable SIZE, you can enter

```
NAME SIZE;
LOCATE 1,LAST XXL;
```

and after the computer responds, enter

```
LOCATE 1,LAST MED;
```

The SAS System again searches the variable SIZE on all observations for the value MED.

STRING command

STRING *variable ...;*
STR *variable ...;*

The STRING command precedes the SEARCH command. It gives a list of character variables that SEARCH should search for matches.

All variables listed **must** be character variables. If no variable name is given, the command displays the names of all the current STRING variables.

The STRING command can be reissued as often as necessary. It does not have to be reissued between uses of any of the searching commands. For example, if you want to find two particular character strings in the variable PLACE, you could use the statements

```
STRING PLACE;
SEARCH ALL 1,LAST BURGH;
```

and after SAS responds, enter

```
SEARCH ALL 1,LAST BERG;
```

SAS again searches the variable PLACE in all observations for the string BERG.

You can also use variable lists with the STRING command. For example,

```
STRING A-CHAR-B;
```

VERIFY command

VERIFY *option;*
VER *option;*
V *option;*

The VERIFY command allows you to monitor selectively the action taken by change commands by displaying or not displaying changes according to the option that is specified. The option can be ON, OFF, RESET, LIST, or NOLIST.

- If VERIFY ON is specified, all changes made to the data set through the use of the change commands are displayed.
- VERIFY OFF turns off this facility. VERIFY OFF is the default when you invoke the EDITOR procedure.
- VERIFY RESET indicates that the NOGO switch is to be turned off. The NOGO switch is turned on if a searching command with the VERIFY option does not find a match (see **Searching Commands**), and it prevents EDITOR from executing REPLACE and DELETE commands.
- VERIFY LIST displays the contents of an observation after a FIND/SEARCH/LOCATE finds a match for that observation. All variables are listed as if you had entered a LIST command with no variable names.
- VERIFY NOLIST turns off this facility. (VERIFY NOLIST is the default when you invoke PROC EDITOR.)

Do not confuse the VERIFY command with the VERIFY option used with searching commands.

END command

END;
E;
QUIT;
Q;

The END command exits from the EDITOR procedure. It has no operands. A PROC, DATA, or RUN statement, or an end-of-file occurrence also causes the EDITOR procedure to exit.

Display command A display command displays information. LIST is the only display command; it is explained below.

LIST command

LIST *range variable ...;*
L *range variable ...;*

The LIST command lists the values of one or more specified variables in a range of observations.

If the range is not specified, only the current observation is listed. If no variable is specified, all variables are listed.

If more than one variable is given, the variables indicated are listed in the order specified. Suppose variables A, B, and C have the values 1, 2, and 3, respectively, and that the current observation is 1. The command

```
LIST;
```

displays

```
1 A=1 B=2 C=3 ;
```

The command

```
LIST C A;
```

displays

```
1  C=3 A=1 ;
```

The command

```
LIST 1 B;
```

displays

```
1 B=2 ;
```

LIST always gives the observation number and ends the list with a semicolon. If formats have been specified for the variables, the formatted values are listed. (Remember the format width restrictions for PROC EDITOR discussed in **Format Statement**.)

See the description of the REPLACE command to learn how the LIST command is used in conjunction with the REPLACE command.

Change commands The change commands make modifications to the existing SAS data set. These commands are REPLACE, ADD, DELETE, and DUP. Each is explained below.

If the NOGO switch is turned on by a searching command that did not find a match, the REPLACE AND DELETE commands do not perform the change. The NOGO switch is reset (turned off) by the VERIFY command (see above).

REPLACE command

REPLACE *options range variable1=value1 variable2=value2 ...;*
REP *options range variable1=value1 variable2=value2 ...;*
R *options range variable1=value1 variable2=value2 ...;*

The REPLACE command changes one or more values of specified variables in a range of observations.

For example, the statement

```
REPLACE  1,5  YEAR=1985  COST=2000;
```

causes the value of YEAR to be changed to 1985 and the value of COST to be changed to 2000 for observations 1 through 5.

If the VERIFY ON command has been given, the variables to be changed are displayed before and after changing. If formats have been specified for the variables, the formatted values are displayed. (Remember the PROC EDITOR format width restrictions discussed in **Format Statement**.)

ADD command

ADD *variable1=value1 variable2=value2 ...;*
A *variable1=value1 variable2=value2 ...;*

The ADD command adds a new observation to the end of a data set. The statement

```
ADD X=3.5  Y=2.0  Z=0.7;
```

adds one observation to the data set.

The ADD command does not use a range specification because it only adds one observation at a time. To add several observations with the same value, enter the ADD command once, then enter the DUP command as many times as necessary to add the observation.

DELETE command

DELETE *range;*
DEL *range;*
D *range;*

The DELETE command sets all the variables in a specified range of observations to missing. All numeric variables are set to "." and all character variables are set to blank. The DELETE command does not entirely delete the observation in the way the DATA step's DELETE statement does. In order to fully delete the observation, you must process the data set with a subsequent DATA step.

DUP command

DUP *range;*

The DUP command is used to duplicate observations in a specified range in the SAS data set. For example, the statement

```
DUP 1,3;
```

duplicates observations 1 through 3 and adds the copies to the end of the SAS data set.

As another example, suppose you want an observation duplicated 20 times. The observation to be copied is observation 50. Enter:

```
DUP 50;          copy the observation
L;               determine the observation number (say,100)
DUP 100;         make a second copy
DUP 100,LAST;    copy both copies for 4 copies
DUP 100,LAST;    copy all copies for 8 copies
DUP 100,LAST;    copy all copies for 16 copies
DUP 100,103;     make 4 copies for a total of 20
```

Positioning commands PROC EDITOR uses an observation pointer that points to the current observation. At the beginning of an EDITOR session, the pointer moves to the first observation. When an observation number is specified in an

EDITOR command the pointer is moved to that observation. For example, specifying

```
LIST 14;
```

moves the pointer to observation 14 and lists the values in observation 14. When a range of observations is specified, the pointer is moved to the first observation in the range, and the command is executed for the observation; next the pointer is moved to the second observation in the range, and so on.

The pointer remains at the observation for which a command has just been executed until it is repositioned by one of the positioning commands or by a range specification in another EDITOR command. (Using a range specification in a command has the effect of repeating the command for each observation in the range.)

You can use the positioning commands to reposition the pointer at any time in an EDITOR session.

TOP command

```
TOP;
```

The TOP command moves the pointer to the first observation.

BOTTOM command

```
BOTTOM;
```

The BOTTOM command moves the pointer to the last observation.

UP command

```
UP n;
```

The UP command moves the pointer up *n* observations. If the pointer is at observation 23, specifying

```
UP 3;
```

moves the pointer to observation 20.

DOWN command

```
DOWN n;
```

The DOWN command moves the pointer down *n* observations. If the pointer is at observation 90, specifying

```
DOWN 10;
```

moves the pointer to observation 100.

DETAILS

Repeating FIND, LOCATE, or SEARCH Commands

With searching commands it is often convenient to find an occurrence based on a value list, look more closely at the observation, then continue the search for occurrences in other observations. If the value list is long, it is inconvenient to reenter it and calculate the range of observations. Therefore, a repeat facility is available under PROC EDITOR.

To repeat a FIND, LOCATE, or SEARCH command, reenter the command **without** the value list, and the previous value list is used. A range can be specified, but if no range is specified, the default range is *m,n* where *m* equals the current obser-

vation plus one, and *n* is the same as the last observation of the previously indicated range. For example,

```
FIND 1,10 X=1;  (found at observation 3)
FIND;           (interpreted as FIND 4,10 X=1; found at 6)
FIND 20,LAST;   (interpreted as FIND 20,LAST X=1; found at 30)
FIND;           (interpreted as FIND 31,LAST X=1;)
```

To use the repeat facility, the new searching command must be the same as the previous searching command. That is, you cannot enter a LOCATE command with a value list, then enter a FIND command without a value list.

Using PROC EDITOR in Batch Mode

Although the EDITOR procedure is designed for interactive use, it can be used in batch mode to make changes to existing SAS data sets.

To use PROC EDITOR in batch, you need the SAS statements

```
PROC EDITOR DATA=SASdataset;
RUN;
```

where *SASdataset* is the name of the SAS data set to be edited.

Follow the PROC EDITOR and RUN statements with the EDITOR commands needed to make the desired changes in the data set.

PROC EDITOR's responses to the commands are printed on the SAS log.

In batch mode, the VERIFY option should always be used with the FIND command to guard against inadvertently changing or deleting the wrong values or observations.

For example, suppose you wanted to find the observation where the value for NAME was JOHN DOE and then change the value of the variable CITY in that observation to CHICAGO. In a batch environment, you would use these statements:

```
PROC EDITOR DATA=MY.NAMEFILE;
   FIND VER 1,LAST NAME='JOHN DOE';
   REP CITY=CHICAGO;
```

The VER option in the FIND command protects against changing the CITY value in the last observation of the data set if no observation with a NAME value of JOHN DOE is found. (Recall that when a searching command does not find an observation with the specified value, it stops at the last observation in the range specified.)

When a search fails in batch mode, the NOGO switch is turned on, which means that EDITOR stops executing REPLACE and DELETE commands. To reset the NOGO switch so that PROC EDITOR begins executing these commands again, use the statement

```
VERIFY RESET;
```

To protect further against mistakes, EDITOR stops executing REPLACE, DELETE, and ADD commands if a syntax error is encountered in the EDITOR commands.

Output Data Set

The output from PROC EDITOR is the edited version of the SAS data set used for input. There is no printed output.

EXAMPLES

Editing a SAS Data Set: Example 1

The example session below shows the data set ECONOMIC before editing, a PROC EDITOR interactive session, and the edited data set.[2]

```
1? proc print data=economic;run;

OBS    COUNTRY        AGRICULT  INDUSTRY  CONSTRUC  TRADE  TRANSCOM
1      AFGHANISTAN      49        17        8        12       3
2      BANGLADESH       54         8        5         9       6
3      BOLIVIA          18        25        4        19       8
4      BURMA            47        10        1        29       3
5      CHILE            10        27        2        29       4
6      ECUADOR          20        29        6        13       5
7      EGYPT            24        23        4        11       7
8      ETHIOPIA         44        11        4         9       5
9      GHANA            51        14        5        13       4
10     GUATEMALA        28        14        2        32       4
11     HAITI            41        15        4        10       2
12     INDIA            36        18        5        11       5
13     IRAN              9        48        9         5       3
14     IRAQ              7        63        2         5       4
15     KENYA            34        13        4        10       4
16     MEXICO            9        30        6        31       3
17     NEPAL            67        10        1         5       3
18     NIGERIA          26        38        6        11       3
19     PAKISTAN         31        16        5        13       6
20     PERU             13        36        4        15       7
21     RHODESIA         15        29        3        11       6
22     SUDAN            39        11        4        16       6
23     TURKEY           27        21        5        13       8
24     UGANDA           53        10        2         9       3
25     VENEZUELA         6        39       17        10      11
26     VIETNAM          29         7        1        18       4
27     YEMEN ARAB REP.  35         6        8        22       3
28     YUGOSLAVIA       17        40       10        21       8
29     ZAIRE            19        22        6        16       4
30     ZAMBIA           13        35        9        12       5

2? proc editor data=economic;
WELCOME TO THE SAS LINE EDITOR. BEGIN ENTERING COMMANDS.

3> verify on;
4> find 1,last country=rhodesia;
FOUND AT OBS= 21 .

5> replace 21 country=zimbabwe;
21 COUNTRY=RHODESIA ;
21 COUNTRY=ZIMBABWE ;

6> add country=kampuchea agricult=60 industry=10 construc=18
   trade=10 transcom=2;
   31 COUNTRY=KAMPUCHEA AGRICULT=60 INDUSTRY=10 CONSTRUC=18 TRADE=10
```

```
    TRANSCOM=2 ;

7> name agricult;
8> locate all 1,last 50;
NOT FOUND. OBS= 31 .

9> find all 1,last agricult>60;
FOUND AT OBS= 17 .
END OF SEARCH. OBS= 31 .

10> delete 28;
     28 COUNTRY=YUGOSLAVIA AGRICULT=17 INDUSTRY=40 CONSTRUC=10
       TRADE=21 TRANSCOM=8 ;
     28 COUNTRY= AGRICULT=. INDUSTRY=. CONSTRUC=. TRADE=. TRANSCOM=. ;

11> end;
EXIT FROM SAS EDITOR.

12? proc print;run;
```

OBS	COUNTRY	AGRICULT	INDUSTRY	CONSTRUC	TRADE	TRANSCOM
1	AFGHANISTAN	49	17	8	12	3
2	BANGLADESH	54	8	5	9	6
3	BOLIVIA	18	25	4	19	8
4	BURMA	47	10	1	29	3
5	CHILE	10	27	2	29	4
6	ECUADOR	20	29	6	13	5
7	EGYPT	24	23	4	11	7
8	ETHIOPIA	44	11	4	9	5
9	GHANA	51	14	5	13	4
10	GUATEMALA	28	14	2	32	4
11	HAITI	41	15	4	10	2
12	INDIA	36	18	5	11	5
13	IRAN	9	48	9	5	3
14	IRAQ	7	63	2	5	4
15	KENYA	34	13	4	10	4
16	MEXICO	9	30	6	31	3
17	NEPAL	67	10	1	5	3
18	NIGERIA	26	38	6	11	3
19	PAKISTAN	31	16	5	13	6
20	PERU	13	36	4	15	7
21	ZIMBABWE	15	29	3	11	6
22	SUDAN	39	11	4	16	6
23	TURKEY	27	21	5	13	8
24	UGANDA	53	10	2	9	3
25	VENEZUELA	6	39	17	10	11
26	VIETNAM	29	7	1	18	4
27	YEMEN ARAB REP.	35	6	8	22	3
28		.	.	.	.	.
29	ZAIRE	19	22	6	16	4
30	ZAMBIA	13	35	9	12	5
31	KAMPUCHEA	60	10	18	10	2

Using a Macro with PROC EDITOR: Example 2

The following example shows how you can use the SAS macro language within PROC EDITOR (see "SAS Macro Language"). A data set contains the variables AGE, BENEFIT, SEX, and DEDUCT. You want to find all observations in which AGE>=65, BENEFIT=2, and DEDUCT<100, and change the BENEFIT value to 3 and the DEDUCT value to 100. A macro issues a REPLACE command to change the BENEFIT and DEDUCT values, and then issues a FIND command. Notice that the macro does not have to repeat the entire FIND command; instead, it takes advantage of the "repeat find" facility, repeating the initial FIND command in line 7.

```
1? proc print data=ssdata;run;

OBS   AGE  BENEFIT  SEX  DEDUCT
 1     59     2      1     95
 2     66     2      1     50
 3     72     3      2    100
 4     58     1      2     75
 5     68     2      1     80
 6     48     2      1     95
 7     65     2      1     50
 8     72     3      2    100
 9     58     1      2     75
10     68     2      1     80
11     55     1      1     20

2? proc editor data=ssdata;

WELCOME TO THE SAS LINE EDITOR. BEGIN ENTERING COMMANDS.

3> %macro doit;
4> replace benefit=3 deduct=100;
5> find;
6> %mend doit;
7> find 1,last age>=65 benefit=2 deduct<100;
FOUND AT OBS= 2 .

8> %doit;
FOUND AT OBS= 5 .

9> %doit;
FOUND AT OBS= 5 .

10> %doit;
FOUND AT OBS= 10 .

11> %doit;
NOT FOUND. OBS= 11 .

12> end;
EXIT FROM SAS EDITOR.

13? proc print data=ssdata;run;
```

OBS	AGE	BENEFIT	SEX	DEDUCT
1	59	2	1	95
2	66	3	1	100
3	72	3	2	100
4	58	1	2	75
5	68	3	1	100
6	48	2	1	95
7	65	3	1	100
8	72	3	2	100
9	58	1	2	75
10	68	3	1	100
11	55	1	1	20

NOTES

1. **VMS:** This is not a problem under VMS because you can specify that more than one version of your file be retained.

2. **VSE:** Under ICCF, the line number prompts do not appear. Instead, ICCF prompts you for further commands by displaying the message "*enter data?"

Chapter 35

The FORMAT Procedure

Operating systems: All

ABSTRACT

Use the FORMAT procedure to define your own output formats for character or numeric values. The new formats can be associated with variables in:

- DATA step PUT or FORMAT statements
- a PROC step with a FORMAT statement.

See "SAS Informats and Formats" for general information on how informats and formats are used with the SAS System. Also see the **FORMAT Statement**.

INTRODUCTION

You can define formats to your own specifications with the FORMAT procedure. PROC FORMAT produces two kinds of formats:

- Value-labeling formats associate labels with values. The VALUE statement generates these formats. For example, you might store SEX as a numeric code, 1 or 2, but print it as a character value using the following statement:

```
VALUE SEX 1='MALE' 2='FEMALE';
```

  Both character and numeric variables can have value-labeling formats.
- Picture formats specify templates for printing numbers, giving specifics such as leading zeros, comma and decimal punctuation, fill characters, prefixes, and negative number representation. For example, you might want to define a picture format for phone numbers with the following statement:

```
PICTURE PHONENUM LOW-HIGH='000/000-0000';
```

  Picture formats are for numeric variables only.

Background

PROC FORMAT evaluates and processes each format you define and stores it as a member of a library. The formats you create are temporary formats if stored in a temporary library, or they can be permanent if saved in a permanent library. By default, formats are temporary (stored in a temporary library). If you want to store formats permanently, use the LIBRARY= option of the PROC FORMAT statement described later in this chapter. More details on storing formats are also provided later in this chapter.

Formats, whether standard SAS formats or those created with PROC FORMAT, are used in PUT and FORMAT statements in a DATA step and in various procedures in a FORMAT statement. If you use a format in a PROC step, the procedure then uses the format to print the variable's values. For example:

```
PROC PRINT;
   VAR PHONE;
   FORMAT PHONE PHONENUM.;
```

The values of the variable PHONE are printed according to the specifications of the PHONENUM. format. Note that in some cases variables with formats are classified according to formatted values rather than internal values.

Format names, whether standard SAS formats or those created with PROC FORMAT, **must** end with a period. User-written formats can have width and decimal specifications like SAS-provided formats (for example, DOLLAR10.2). Formats with width and decimal specifications do not need an additional period after the format name; the decimal specification is sufficient, as it is in SAS-provided formats. A decimal specification is ignored in picture formats.

Do not confuse the FORMAT procedure with the FORMAT statement. PROC FORMAT creates user-defined formats. The FORMAT statement associates an existing format (either standard SAS or user-defined) with one or more variables. The FORMAT statement can be used in either a DATA or a PROC step. When used in the DATA step that creates a SAS data set, the FORMAT statement associates the format permanently with a variable. The FORMAT statement in a PROC step associates the format with a variable only for the duration of that step.

Note: permanently associating a format with a variable does not permanently **store** the format. Serious complications arise if you do **not** save the format for

a permanent data set variable. Be sure you permanently store all formats associated with permanent data set variables. See the section **Creating Temporary and Permanent Formats** at the end of this chapter for more information.

The following example uses PROC FORMAT to define picture formats and then uses the formats with a PUT statement and PROC PRINT.

```
PROC FORMAT;
   PICTURE PHONENUM OTHER='000/000-0000';
   PICTURE SSNUM OTHER='999-99-9999';

DATA A;
   INPUT PHONE SS;
   PUT PHONE PHONENUM.;
   FORMAT SS SSNUM.;
   CARDS;
9194678000 333221111
9198344381 555667777
;
PROC PRINT;
   FORMAT PHONE PHONENUM.;
```

The results of the PUT statement are printed on the log:

```
919/467-8000
919/834-4381
```

The FORMAT statement in the PROC PRINT step indicates that the procedure is to use the format PHONENUM. to print PHONE values. Because the SSNUM. format is associated with the SS variable by a FORMAT statement within a DATA STEP, the association is permanent. The PRINT procedure prints both SS and PHONE values with the picture formats created for them by PROC FORMAT. The output from PROC PRINT looks like this:

```
 OBS      PHONE          SS

  1   919/467-8000  333-22-1111
  2   919/834-4381  555-66-7777
```

The PHONE values print on the log in the pattern specified by PHONENUM. However, the PHONENUM. format is not permanently associated with PHONE the way SSNUM. is associated with SS. This FORMAT statement in the DATA step associates the PHONENUM. format with the PHONE variable permanently:

```
FORMAT SS SSNUM. PHONE PHONENUM.;
```

Value-labeling formats are especially useful when data are stored with uninformative values. For example, suppose you are analyzing the results of a questionnaire. One variable in your data set is PARTY, which represents the political party of the respondent. PARTY'S values are 1 for DEMOCRAT, 2 for REPUBLICAN, 3 for INDEPENDENT, and 4 for OTHER. Whenever values of the variable PARTY are printed, you want the name of the party substituted for the corresponding number. PROC FORMAT gives the appropriate directions for printing the party names. These SAS statements read the political questionnaire data, define a format called P., associate the P. format with the PARTY variable, and then produce a crosstabulation table of AGE by PARTY:

```
DATA POLIT;
   INPUT NAME $ ID AGE PARTY;
   CARDS;
```

```
JOHN  191 18 1
JOYCE 218 19 2
HARRY 923 21 4
JAMES 432 20 3
;
PROC FORMAT;
   VALUE P 1='DEMOCRAT'
           2='REPUBLICAN'
           3='INDEPENDENT'
           4='OTHER';
PROC PRINT;
   TITLE 'WITHOUT FORMAT';
PROC PRINT;
   TITLE 'WITH FORMAT';
   FORMAT PARTY P.;
PROC FREQ;
   TABLES PARTY*AGE;
   FORMAT PARTY P.;
```

For another example of the FORMAT procedure, see the chapter, "PROC Step Applications."

SPECIFICATIONS

The FORMAT procedure is controlled by the following statements:

PROC FORMAT *options*;
  VALUE *name (options)*
      range=label
      ...;
  PICTURE *name (options)*
      range=picture (options)
      ...;

Each new VALUE or PICTURE statement defines a format. You can specify as many formats as you want in a PROC FORMAT step, with one format per VALUE or PICTURE statement.

PROC FORMAT Statement

PROC FORMAT *options;*

The following options can be used in the PROC FORMAT statement:

LIBRARY=*libref* | *fileref*
: specifies a libref or fileref that indicates where the format is to be stored.[1] See the appendix to this chapter for more details on storing formats for your operating system.

  Operating systems: All

DECK
: specifies that the object deck produced by PROC FORMAT is to be written to the VSE SYSPCH logical unit. This option **must** be specified to create permanent formats under VSE. (See **Creating Temporary and Permanent Formats** at the end of this chapter.)

  Operating system: VSE

VALUE Statement

VALUE *name (options)*
range1=label1
range2=label2
... ;

The elements of the VALUE statement are

name names the format being created. You **must** specify a name in the VALUE statement. The name must be a valid SAS name up to eight characters long, not ending in a number. If the format is for character variables, the first character must be a dollar sign ($), while the remaining seven characters must follow the rules for a valid SAS name. The name must not be a reserved SAS word.[2] Refer to the format later by using the name followed by a period. (Do **not** put a period after the format name in the VALUE statement.)

options are enclosed in parentheses and follow the format name. Valid options are MAX=, MIN=, DEFAULT=, and FUZZ=. They are described in detail below.

ranges specifies a range of values, a list of values, or a list of ranges. You **must** specify one or more ranges in the VALUE statement. The syntax is

value	(single values)
value-value	(a range of values)
range,range	(list of ranges or values)

Do not overlap the values in ranges or value lists.

The appropriate label is the one associated with the range in which the character or number falls. For example:

```
VALUE ABC 1='A' 2='B' 3='C';
VALUE AGEFMT 0-12='CHILD'
             13-19='TEEN'
             20-HIGH='ADULT';
VALUE SEXFMT 1='FEMALE'
             2='MALE'
             0,3-9='MISCODED';
```

The first example uses single values, the second uses ranges, and the third uses both single values and ranges.

A noninclusive range can be specified. For example,

```
VALUE XXX 1-2='X1' 2-3='X2';
```

is confusing because the value 2 could be part of the first or second range. You can specify

```
VALUE XXX 1-<2='X1' 2-3='X2';
```

and the value 2 goes only with the second range. The notation

```
number1 <- number2
```

indicates that the first number is not part of the range. The notation

```
number1 -< number2
```

indicates that the second number is not part of the range. Also, if a range such as

```
VALUE XXX 1-2='X1' 2-3='X2';
```

is specified, the range is automatically changed to

```
VALUE XXX 1-2='X1' 2<-3='X2';
```

to avoid confusion. The first occurrence of a value is always chosen to be the inclusive one.

Formats may include the keywords LOW and HIGH in a range specification. In the AGEFMT. example above, the range 20-HIGH refers to all values from 20 through the largest value of the variable. (The LOW keyword does not include missing values.)

The keyword OTHER may be used on the left side of the equal sign to mean all values not given in any other range or value specification for the format. In the SEXFMT. example above, OTHER=MISCODED could be substituted for 0,3-9=MISCODED to mean that all values other than 1 and 2 are to have a printed value of MISCODED:

```
VALUE SEXFMT 1='FEMALE'
             2='MALE'
         OTHER='MISCODED';
```

labels

Character values must be enclosed in single quotes.[3] value labels can be up to forty characters. (However, some procedures use only the first eight or sixteen characters of a label.) All labels must be in quotes.[4] For example,

```
VALUE XXX 1='YES' 2='NO';
VALUE $XXX 'A'='GOOD' 'B'='BAD';
```

If a label contains a single quote, write it as two separate single quotes:

```
VALUE SECT 1='SMITH''S CLASS'
           2='DOE''S CLASS';
```

If a VALUE statement does not include all of a variable's values, those in the VALUE statement are printed with the specified format, while other values are printed with a default format. For example, these statements

```
PROC FORMAT;
   VALUE TEMP 98.6='NORMAL';
DATA A;
   INPUT T;
   PUT T TEMP6.1;
   CARDS;
98.4
98.6
101.2
```

```
;
```

produce these lines

```
  98.4
NORMAL
 101.2
```

Note: a numeric value with a default format is right-aligned in a field with a width equal to that given in the PUT statement. If a field width is not given, the default width of the format is used. Character values using a default format are left-aligned.

PICTURE Statement

PICTURE *name (options)*
range1=picture1 (options)
range2=picture2 (options)
... ;

The elements of the PICTURE statement are listed below.

name names the new format. You **must** specify a name in the PICTURE statement. The name must be a valid SAS name up to eight characters long, not ending in a number. The name cannot be a reserved SAS word; if it is, the SAS System issues a message.[5] Refer to the format later by using the name followed by a period. (Do **not** put a period after the format name in the PICTURE statement.)

options are enclosed in parentheses and follow the format name. The valid options are MAX=, MIN=, DEFAULT=, and FUZZ=. They are described in detail below.

ranges specifies a range of values, a list of values, or a list of ranges. The syntax is

value	(single values)
value-value	(a range of values)
range,range	(list of ranges or values)

The appropriate picture is the one associated with the range in which the value falls. For example:

```
PICTURE FM LOW -< 0 = '99999-'
           0 -HIGH = '99999+';
```

LOW and HIGH are keywords representing the largest negative and positive numbers, respectively. The '−< 0' indicates that the LOW range is **not** to include zero. The special range OTHER can be used to collect numbers not included in any other range. Missing values are represented with a period (.).

pictures specifies a template for printing numbers. The picture is a sequence of characters in single quotes. The maximum length for a picture is twenty-four characters.

Pictures are specified with two types of characters: digit selectors and message characters.

Digit selectors are characters that define positions for numeric values. The digit selectors can be either 0, or the numbers 1 through 9. If the picture is specified with zeros, and the number to be formatted contains leading zeros, the leading zeros are not formatted into the field. To print leading zeros, you may use any digit 1 through 9 to define the positions. All digits, even leading zeros, are formatted. The picture cannot begin with a message character. (See the PREFIX and FILL options.)

Message characters are non-numeric characters that are printed just as they appear in the picture. They are inserted into the picture after the numeric digits are formatted. For example, this PICTURE statement:

```
PICTURE DAY 01-31='00';
```

causes the value 02 to be printed like this:

```
2
```

This PICTURE statement:

```
PICTURE DAY 01-31='99'
            OTHER='99-ILLEGAL DAY VALUE';
```

prints the values 02 and 67 as

```
                    02
67-ILLEGAL DAY VALUE
```

In this PICTURE statement, the characters 99 are digit selectors, and the characters -ILLEGAL DAY VALUE are message characters.

options are enclosed in parentheses and specify fill characters, prefix characters, and multipliers, described below. This field is optional.

PICTURE Options

These options can be specified for each picture in a picture format.

FILL='*character*' specifies a fill character. This character replaces the leading characters of the picture until a significant digit is encountered. The default is FILL=' ' (blank). The specified character must be enclosed in single quotes. One use of the FILL= option is to prevent a number with leading blanks from being altered, for example, on a check. (See the example below, following the PREFIX description.)

PREFIX='*character*' is a one- or two-character prefix placed in front of the first significant digit of the value. The default is no prefix. The specified character must be enclosed in single quotes. The prefix occupies positions within the picture, not additional positions. Thus, if the picture is not wide enough to contain both the value and the prefix, the prefix is truncated or omitted. The PREFIX=

option is often used for leading dollar signs and minus signs. For example, the picture

```
PICTURE PAY OTHER='00,000,000.00'
   (FILL='*' PREFIX='$');
```

prints the value 25500 as

```
***$25,500.00
```

MULTIPLIER=*n*
MULT=*n*
specifies a number to be multiplied by a value before it is formatted. The main use of the MULT= option is to get a data value containing decimal points into a form that will fit into the picture template correctly since picture format decimal points are part of the template, not part of the data value. For example, to have 1600000 print as $1.6M use these PREFIX= and MULTIPLIER= specifications:

```
PICTURE MILLION LOW-HIGH='00.0M'
   (PREFIX='$' MULT=.00001);
```

If you do not specify a value for MULT=, the procedure uses a default value of 10^n, where *n* is the number of digits after the first decimal point in the picture. For example, suppose your data contain a value 123.456 and you want to print it using a picture of '999.999'. FORMAT multiplies 123.456 by 10^3 to obtain a value of 123456; it then fits this value into the picture template to produce 123.456.

NOEDIT
forces the format to treat the picture as a value label rather than as a picture specification. This means that numbers are message characters rather than digit selectors; that is, the numbers in the picture are printed as they appear. For example, the statements:

```
PROC FORMAT;
   PICTURE MILES 1-99='000000'
       100-HIGH='>100 MILES' (NOEDIT);

DATA TEMP;
   INPUT NAME $ DISTANCE 3.;
   CARDS;
JOHN 300
MARY 600
DAVID 27
ANN 2
;
PROC PRINT;
   FORMAT DISTANCE MILES.;
```

produce this output:

```
OBS  NAME    DISTANCE
 1   JOHN    >100 MILES
 2   MARY    >100 MILES
 3   DAVID           27
 4   ANN              2
```

Format Options (for VALUE or PICTURE)

A format is stored with width attribute information. You may find occasions that need one of the following format attributes:

MIN=*n* specifies a minimum width for the format. If MIN= is not given, the length of the longest picture or value label is the minimum default width.

MAX=*n* specifies a maximum width for the format. This can be larger than the picture or value label since picture formats can pad with fill characters on the left. The maximum MAX= width allowed is 40. If MAX= is not given, the default is 40.

DEFAULT=*n* specifies the default width if the format does not have a width specification. The default value for DEFAULT is the length of the longest picture or value.

FUZZ=*n* specifies a fuzz factor. If a number does not match a value or fall in a range exactly, but comes within the fuzz value, it is considered a match. For example:

```
PROC FORMAT;
   VALUE ABC (FUZZ=.5) 1='A' 2='B' 3='C';
```

FUZZ=.5 means that if a variable value falls within .5 of a value specified in the VALUE statement, the corresponding label is used to print the value. So using the ABC. format to print a value of 1.2 produces an A, the value 1.7 produces a B, and the value 2.8 produces a C.

Only VALUE formats use FUZZ=; PICTURE formats do not.

DETAILS

Character Data Limitations

The VALUE statement of PROC FORMAT processes character values with a maximum of sixteen characters. If a value has more than sixteen characters, the procedure truncates the value and processes the first sixteen characters. If more than one character value in a VALUE statement has the same first sixteen characters, the SAS System issues an error message indicating that the value is a duplicate. The labels specified by a VALUE statement can have a maximum of forty characters.

PICTURE Logic

The SAS System uses the following logic when printing a value with a PICTURE format. For illustration, suppose that you define this format

```
PICTURE MILLION LOW-HIGH = '00.0M'
   (PREFIX='$' MULT=.00001);
```

and you are formatting the value 23468977.

1. Locate the picture associated with the corresponding range for the number to be formatted. If there is no range for the number, use the OTHER specification. If there is no OTHER specification, use the standard SAS numeric format.

2. If the picture contains only message characters (that is, it does not have digit selectors), or has the NOEDIT option, then output the label and ignore the rest of the steps.
3. Multiply the number by the multiplier. In our example, the multiplier is .00001, giving a result of 234.68977.
4. Take the absolute value, that is, ignore the sign.
5. Truncate the value to an integer. If the number is within 10^{-8} of a higher integer, round up. In other words, if the result is between 1 and .00000001, round up to the next integer value and truncate. In our example, the rounded and truncated value is 235.
6. Convert the number to a string of digits, the "source string." That is, convert the number from floating point to character string representation.
7. Point to the digit corresponding to the high-order (left-most) digit selector in the picture. If the number is too large to fit into the picture, ignore the high-order digits, that is, truncate on the left. This has the effect of converting from a floating point number to a standard number. There is no error condition for this "overflow," although you can use special error ranges to control it.
8. Turn the significance switch off. The significance switch is an indicator of whether or not a digit is significant.

Now the format starts scanning the picture and source strings.

1. With a message (non-numeric) picture character, when the significance switch is off, the fill character is put into the result. If the significance switch is on, the message character is placed in the result.
2. With a digit selector (numeric) picture character, a digit is retrieved from the source string. When either the picture character is nonzero or the source string digit is nonzero, the significance switch is turned on. If the significance switch is on, the source digit is placed in the result. Otherwise, when the picture character and the source string digit are zero, the fill character is placed in the result.

   In our example, the picture character is a zero digit selector, and the source string digit is nonzero. Therefore, the significance switch is on, and the source digit is placed in the result.
3. Now the number is edited. Prefix characters are placed before the first nonfill character in the result. If the format width is larger than the picture length, the result is padded on the left with one fill character and the rest blanks. If the format width is smaller than the picture, the result is truncated on the left.

   In our example, the prefix character '$' is placed before the first digit, and the message character 'M' is the last character in the result:

```
$23.5M
```

Permanent Formats

When you create a permanent SAS data set and assign user-written formats to any of its variables, be sure to store the formats permanently. Serious complications arise if you do **not** save the format for a permanent data set variable. The data set directory marks the variable as having a format, but when the variable is referenced in later jobs, the SAS System cannot find that format because it has not been saved. If the global option FMTERR is in effect, a "format not found" SAS error message is issued. If the global option NOFMTERR is in effect, the variable is processed with a default format; however, the information contained in the format is lost.

Directions for creating temporary and permanent formats depend on the operating system. Refer to the appropriate endnote for directions that apply to your operating system.

Printing the Contents of Format Libraries

The FORMAT procedure has no facility for printing formats once the formats have been created. The SAS supplemental procedure FMTLIB can both print and output the contents of a format library. See the *SAS Supplemental Library User's Guide* for documentation.

EXAMPLES

The VALUE Statement Used with PROC FORMAT: Example 1

The following three format definitions are useful for transforming state codes or abbreviations into the state names. The $ZIPST. format corresponds to the ZIPNAME function; $STATE. to STNAME; and STATE. to FIPNAME. These formats are useful because you can access them in a PROC step, whereas you must use a DATA step to use the state functions. In addition, you can use these formats with the PUT function in the DATA step to create variables. For example, in this DATA step:

```
DATA NEW;
   INPUT ZIP $ ABBREV $ CENSUSBU;
   STATE1=PUT(ZIP,$ZIPST.);
   STATE2=PUT(ABBREV,$STATE.);
   STATE3=PUT(CENSUSBU, STATE.);
   CARDS;
27511 NC 37
;
```

STATE1, STATE2, and STATE3 all have a value of NORTH CAROLINA. See the chapter "DATA Step Applications" for more information on creating variables with the PUT function. Also see the chapter "PROC Step Applications" for more examples using PROC FORMAT.

Because many users may use these formats frequently, they can be generated and stored by your installation in the SAS load library to be available for general use.

```
PROC FORMAT PRINT;

   *-----$ZIPST: CONVERTS CHARACTER ZIP CODES TO STATE NAMES-----;
   VALUE $ZIPST
      00600-00999='PUERTO RICO'     01000-02799='MASSACHUSETTS'
      02800-02999='RHODE ISLAND'    03000-03899='NEW HAMPSHIRE'
      03900-04999='MAINE'           05000-05999='VERMONT'
      06000-06999='CONNECTICUT'     07000-08999='NEW JERSEY'
      09000-14999='NEW YORK'        15000-19699='PENNSYLVANIA'
      19700-19999='DELAWARE'        20000-20599='DISTRICT OF COLUMBIA'
      20600-21999='MARYLAND'        22000-24699='VIRGINIA'
      24700-26899='WEST VIRGINIA'   27000-28999='NORTH CAROLINA'
      29000-29999='SOUTH CAROLINA'  30000-31999='GEORGIA'
      32000-33999='FLORIDA'         35000-36999='ALABAMA'
      37000-38599='TENNESSEE'       38600-39799='MISSISSIPPI'
      40000-42799='KENTUCKY'        43000-45899='OHIO'
```

```
    46000-47999='INDIANA'          48000-49999='MICHIGAN'
    50000-52899='IOWA'             53000-54999='WISCONSIN'
    55000-56799='MINNESOTA'        57000-57799='SOUTH DAKOTA'
    58000-58899='NORTH DAKOTA'     59000-59999='MONTANA'
    60000-62999='ILLINOIS'         63000-65899='MISSOURI'
    66000-67999='KANSAS'           68000-69399='NEBRASKA'
    70000-71499='LOUISIANA'        71600-72999='ARKANSAS'
    73000-74999='OKLAHOMA'         75000-79999='TEXAS'
    80000-81699='COLORADO'         82000-83199='WYOMING'
    83200-83899='IDAHO'            84000-84799='UTAH'
    85000-86599='ARIZONA'          87000-88499='NEW MEXICO'
    89000-89899='NEVADA'           90000-96699='CALIFORNIA'
    96700-96899='HAWAII'           96900-96999='GUAM'
    97000-97999='OREGON'           98000-99499='WASHINGTON'
    99500-99999='ALASKA'
    ;
*-----$STATE: CONVERTS STATE ABBREV. TO STATE NAMES------;
VALUE $STATE
   'AL'='ALABAMA' 'AR'='ARKANSAS' 'AZ'='ARIZONA' 'CA'='CALIFORNIA'
   'IL'='ILLINOIS' 'KS'='KANSAS' 'CO'='COLORADO' 'ID'='IDAHO'
   'WI'='WISCONSIN' '99'='FOREIGN' 'GA'='GEORGIA' 'NJ'='NEW JERSEY'
   'VA'='VIRGINIA' 'SC'='SOUTH CAROLINA' 'IN'='INDIANA'
   'IA'='IOWA' 'MO'='MISSOURI' 'MD'='MARYLAND' 'NB'='NEBRASKA'
   'DC'='DISTRICT OF COLUMBIA' 'NY'='NEW YORK' 'OH'='OHIO"
   'MN'='MINNESOTA' 'PA'='PENNSYLVANIA' 'TX'='TEXAS' 'FL'='FLORIDA'
   'NC'='NORTH CAROLINA' 'OK'='OKLAHOMA' 'WV'='WEST VIRGINIA'
   'TN'='TENNESSEE' 'DE'='DELAWARE' 'KY'='KENTUCKY'
   'MS'='MISSISSIPPI' 'NM'='NEW MEXICO' 'ND'='NORTH DAKOTA'
   'MA'='MASSACHUSETTS' 'RI'='RHODE ISLAND' 'SD'='SOUTH DAKOTA'
   'LA'='LOUISIANA' 'UT'='UTAH' 'WA'='WASHINGTON'
   'MI'='MICHIGAN'
   ;
*-----STATE: OFFICIAL CENSUS BUREAU CODES FOR STATE-----;
VALUE STATE
   01='ALABAMA'           30='MONTANA'
   02='ALASKA'            31='NEBRASKA'
   04='ARIZONA'           32='NEVADA'
   05='ARKANSAS'          33='NEW HAMPSHIRE'
   06='CALIFORNIA'        34='NEW JERSEY'
   08='COLORADO'          35='NEW MEXICO'
   09='CONNECTICUT'       36='NEW YORK'
   10='DELAWARE'          37='NORTH CAROLINA'
   11='D.C.'              38='NORTH DAKOTA'
   12='FLORIDA'           39='OHIO'
   13='GEORGIA'           40='OKLAHOMA'
   15='HAWAII'            41='OREGON'
   16='IDAHO'             42='PENNSYLVANIA'
   17='ILLINOIS'          44='RHODE ISLAND'
   18='INDIANA'           45='SOUTH CAROLINA'
   19='IOWA'              46='SOUTH DAKOTA'
   20='KANSAS'            47='TENNESSEE'
   21='KENTUCKY'          48='TEXAS'
   22='LOUISIANA'         49='UTAH'
   23='MAINE'             50='VERMONT'
   24='MARYLAND'          51='VIRGINIA'
```

```
25='MASSACHUSETTS'      53='WASHINGTON'
26='MICHIGAN'           54='WEST VIRGINIA'
27='MINNESOTA'          55='WISCONSIN'
28='MISSISSIPPI'        56='WYOMING'
29='MISSOURI'
;
```

Output 35.1 State Code and Abbreviation Formats

```
1       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB X1FORMAT STEP SASBASE  PROC

NOTE: THE JOB X1FORMAT HAS BEEN RUN UNDER RELEASE 5.04 OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
      SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.09 SECONDS.
1          PROC FORMAT PRINT;
2             *-----$ZIPST: CONVERTS CHARACTER ZIP CODES TO STATE NAMES-----;
3             VALUE $ZIPST
4                  00600-00999='PUERTO RICO'    01000-02799='MASSACHUSETTS'
5                  02800-02999='RHODE ISLAND'   03000-03899='NEW HAMPSHIRE'
6                  03900-04999='MAINE'          05000-05999='VERMONT'
7                  06000-06999='CONNECTICUT'    07000-08999='NEW JERSEY'
8                  09000-14999='NEW YORK'       15000-19699='PENNSYLVANIA'
9                  19700-19999='DELAWARE'       20000-20599='DISTRICT OF COLUMBIA'
10                 20600-21999='MARYLAND'       22000-24699='VIRGINIA'
11                 24700-26899='WEST VIRGINIA'  27000-28999='NORTH CAROLINA'
12                 29000-29999='SOUTH CAROLINA' 30000-31999='GEORGIA'
13                 32000-33999='FLORIDA'        35000-36999='ALABAMA'
14                 37000-38599='TENNESSEE'      38600-39799='MISSISSIPPI'
15                 40000-42799='KENTUCKY'       43000-45899='OHIO'
16                 46000-47999='INDIANA'        48000-49999='MICHIGAN'
17                 50000-52899='IOWA'           53000-54999='WISCONSIN'
18                 55000-56799='MINNESOTA'      57000-57799='SOUTH DAKOTA'
19                 58000-58899='NORTH DAKOTA'   59000-59999='MONTANA'
20                 60000-62999='ILLINOIS'       63000-65899='MISSOURI'
21                 66000-67999='KANSAS'         68000-69399='NEBRASKA'
22                 70000-71499='LOUISIANA'      71600-72999='ARKANSAS'
23                 73000-74999='OKLAHOMA'       75000-79999='TEXAS'
NOTE: FORMAT $ZIPST HAS BEEN OUTPUT.
24                 80000-81699='COLORADO'       82000-83199='WYOMING'
25                 83200-83899='IDAHO'          84000-84799='UTAH'
26                 85000-86599='ARIZONA'        87000-88499='NEW MEXICO'
27                 89000-89899='NEVADA'         90000-96699='CALIFORNIA'
28                 96700-96899='HAWAII'         96900-96999='GUAM'
29                 97000-97999='OREGON'         98000-99499='WASHINGTON'
30                 99500-99999='ALASKA' ;
31
32            *-----$STATE: CONVERTS STATE ABBREV. TO STATE NAMES------;
33            VALUE $STATE
34           'AL'='ALABAMA' 'AR'='ARKANSAS' 'AZ'='ARIZONA' 'CA'='CALIFORNIA'
35           'IL'='ILLINOIS' 'KS'='KANSAS' 'CO'='COLORADO' 'ID'='IDAHO'
36           'WI'=WISCONSIN 99='FOREIGN' 'GA'='GEORGIA' 'NJ'='NEW JERSEY'
37           'VA'='VIRGINIA' 'SC'='SOUTH CAROLINA' 'FL'='FLORIDA' 'IN'='INDIANA'
NOTE: FORMAT $STATE HAS BEEN OUTPUT.
38           'IA'='IOWA' 'MO'='MISSOURI' 'MD'='MARYLAND' 'DC'='DISTRICT OF COLUMBIA'
39           'NY'='NEW YORK' 'OH'='OHIO' 'NB'='NEBRASKA' 'MN'='MINNESOTA'
40           'PA'='PENNSYLVANIA' 'TX'='TEXAS' 'NC'='NORTH CAROLINA' 'OK'='OKLAHOMA'
41           'WV'='WEST VIRGINIA' 'TN'='TENNESSEE' 'DE'='DELAWARE' 'KY'='KENTUCKY'
42           'MS'='MISSISSIPPI' 'NM'='NEW MEXICO' 'ND'='NORTH DAKOTA'
43           'MA'='MASSACHUSETTS' 'RI'='RHODE ISLAND' 'SD'='SOUTH DAKOTA'
44           'LA'='LOUISIANA' 'UT'='UTAH' 'WA'='WASHINGTON' 'MI'='MICHIGAN';
45
46            *-----STATE: OFFICIAL CENSUS BUREAU CODES FOR STATE-----;
47            VALUE STATE
48              01='ALABAMA'        02='ALASKA'                    04='ARIZONA'
49              05='ARKANSAS'       06='CALIFORNIA'                08='COLORADO'
```

```
2       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB X1FORMAT STEP SASBASE  PROC
50             09='CONNECTICUT'    10='DELAWARE'       11='D.C.'       12='FLORIDA'
51             13='GEORGIA'                            15='HAWAII'     16='IDAHO'
52             17='ILLINOIS'       18='INDIANA'        19='IOWA'       20='KANSAS'
53             21='KENTUCKY'       22='LOUISIANA'      23='MAINE'      24='MARYLAND'
54             25='MASSACHUSETTS'  26='MICHIGAN'       27='MINNESOTA' 30='MONTANA'
NOTE: FORMAT STATE HAS BEEN OUTPUT.
55             29='MISSOURI'       28='MISSISSIPPI'    31='NEBRASKA'  32='NEVADA'
56             33='NEW HAMPSHIRE'  34='NEW JERSEY'     35='NEW MEXICO' 36='NEW YORK'
57             37='NORTH CAROLINA' 38='NORTH DAKOTA'   39='OHIO'       40='OKLAHOMA'
58             41='OREGON'         42='PENNSYLVANIA'   44='RHODE ISLAND'
59             45='SOUTH CAROLINA' 46='SOUTH DAKOTA'   47='TENNESSEE' 48='TEXAS'
60             49='UTAH'           50='VERMONT'        51='VIRGINIA'
61             53='WASHINGTON'     54='WEST VIRGINIA' 55='WISCONSIN' 56='WYOMING';
NOTE: THE PROCEDURE FORMAT USED 0.54 SECONDS AND 616K.
      THE COMPILE   PHASE USED 0.51 SECONDS.
      THE EXECUTION PHASE USED 0.03 SECONDS.
NOTE: SAS USED 616K MEMORY.
NOTE: THE COMPILE   PHASE USED 0.51 SECONDS.
      THE EXECUTION PHASE USED 0.03 SECONDS.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

The PICTURE Statement Used with PROC FORMAT: Example 2

The ten formats in this example show the various features of picture formats. Note especially the specifications of the format, PROT, that establish error ranges so that numbers that do not fit the picture are flagged with an error message. The last value in the example shows that most of the formats produce a misleading value for an overflowed field.

```
PROC FORMAT;

   PICTURE ACCT    LOW-0   = '000,009.99)' (PREFIX='(')
                   0-HIGH  = '000,009.99 ' ;

   PICTURE PROT    LOW-1E5= '-OVERFLOW'    (NOEDIT)
                   -99999.99-<0= '000,009.99 (PREFIX='-' FILL='*')
                   0-999999.99= '000,009.99' (FILL='*')
                   1E6-HIGH='OVERFLOW'       (NOEDIT);

   PICTURE DOL     LOW-0   = '000,009.99'  (PREFIX='0')
                   0-HIGH  = '000,009.99'  (PREFIX='$');

   PICTURE RSIGN   LOW-0   = '000,009.99-'
                   0-HIGH  = '000,009.00+';

   PICTURE CREDIT  LOW-0   = '00,009.99DR'
                   0-HIGH  = '00,009.99CR';

   PICTURE EUROPE  LOW-0   = '00.009,00'   (PREFIX='-' MULT=100)
                   0-HIGH  = '00.009,00'   (MULT=100);

   PICTURE BLANK   LOW-0   = '000 009.99'  (PREFIX='-')
                   0-HIGH  = '000 009.99';

   PICTURE THOUS   0-HIGH  = '00,009K'     (MULT=.001);

   PICTURE PHONE   LOW-HIGH = '000/000-0000';
```

```
   PICTURE SSNUM   LOW-HIGH = '999-99-9999';
PROC PDS DDNAME=LIBRARY; *---THIS LISTS THE FORMATS IN THE LIBRARY;
```

Output 35.2 Creating and Storing Picture Formats with PROC FORMAT

```
1       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB X2FORMAT STEP SASBASE  PROC

NOTE: THE JOB X2FORMAT HAS BEEN RUN UNDER RELEASE 5.04 OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
      SORT=4
31         PROC PDS DDNAME=LIBRARY; *---THIS LISTS THE FORMATS IN THE LIBRARY;


          DSNAME=SYSABC.DEF00.X2FORMAT.LIBRARY,VOL=SER=(NONE)

          MEMBERS
          ACCT BLANK CREDIT DOL EUROPE PHONE PROT RSIGN SSNUM THOUS


           150 TRACKS ALLOCATED
             2 TRACKS USED
           148 TRACKS UNUSED
             1 EXTENTS
            20 DIRECTORY BLOCKS ALLOCATED
             2 DIRECTORY BLOCKS USED

NOTE: THE PROCEDURE PDS USED 0.10 SECONDS AND 536K.
      THE COMPILE   PHASE USED 0.10 SECONDS.
      THE EXECUTION PHASE USED 0.00 SECONDS.


NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

```
DATA A;
   INPUT X PHONE SSN;
   ACCT=X; PROT=X; DOL=X; RSIGN=X; CREDIT=X; EUROPE=X; BLANK=X; THOU=X;
   FORMAT ACCT ACCT.  PROT PROT.  DOL DOL.  RSIGN RSIGN.
      CREDIT CREDIT.  EUROPE EUROPE.  BLANK BLANK.  THOU THOUS.
      PHONE PHONE.  SSN SSNUM.  X 12.2;
   CARDS;
12345   9194678000 123456789
0       4678000    987654321
-12345  .          .
-187.65 9194678000 111111111
187.65  .          .
.23     .          .
101.23  .          .
1.1E6   .          .
;
PROC PRINT; ID X;
   VAR ACCT PROT DOL RSIGN CREDIT EUROPE BLANK THOU PHONE SSN;
   TITLE 'PRINT THE FORMATS CREATED WITH PROC FORMAT';
```

Output 35.3 Picture Formats Created with PROC FORMAT

PRINT THE FORMATS CREATED WITH PROC FORMAT

X	ACCT	PROT	DOL	RSIGN	CREDIT	EUROPE	BLANK	THOU	PHONE	
12345.00	12,345.00	*12,345.00	$12,345.00	12,345.00+	12,345.00CR	12.345,00	12 345.00	12K	919/467-8000	123-45-
0.00	0.00	******0.00	$0.00	0.00+	0.00CR	0,00	0.00	0K	467-8000	987-65-
-12345.00	(12,345.00)	-12,345.00	-12,345.00	12,345.00-	12,345.00DR	12.345,00	-12 345.00	-12345	.	
-187.65	(187.65)	***-187.65	$-187.65	187.65-	187.65DR	-187,65	-187.65	-187.65	919/467-8000	111-11-
187.65	187.65	****187.65	$187.65	187.65+	187.65CR	187,65	187.65	0K	.	
0.23	0.23	******0.23	$0.23	0.23+	0.23CR	0,23	0.23	0K	.	
101.23	101.23	****101.23	$101.23	101.23+	101.23CR	101,23	101.23	0K	.	
1100000.00	100,000.00	OVERFLOW	100,000.00	100,000.00+	0.00CR	0,00	100 000.00	1,100K	.	

APPENDIX: TEMPORARY AND PERMANENT FORMATS

OS Batch

To create a temporary format under OS batch, use a PROC FORMAT statement without a LIBRARY= option. The format is associated with the variable only for the duration of the PROC step. For example,

```
PROC FORMAT;
   VALUE GRADE 0-69='F' 70-74='D' 75-84='C' 85-94='B' 95-100='A';
```

To store formats permanently under OS batch, use JCL to create a permanent partitioned data set (PDS) for your format library. Use SASLIB as the fileref for the partitioned data set. Specify LIBRARY=SASLIB in the PROC FORMAT statement. The name of the format becomes the member name of the file in the PDS.

This example creates a PDS in OS batch and stores two formats in it:

```
// EXEC SAS
//SASLIB DD DSN=ACCT.ME.STORE,UNIT=SYSDA,VOL=SER=volume,
//          SPACE=(TRK,(4,2,2)),DISP=(NEW,CATLG)
PROC FORMAT LIBRARY=SASLIB;
   PICTURE PHONE OTHER='000/000-0000';
   VALUE P 1='DEMOCRAT'
           2='REPUBLICAN'
           3='INDEPENDENT'
           4='OTHER';
```

To use the stored formats in other jobs, include a DD statement in your JCL referring to the partitioned data set; the fileref of this statement **must** be SASLIB, as in this example:

```
// EXEC SAS
//SASLIB DD DSN=ACCT.ME.STORE,DISP=SHR
DATA POLIT;
   INPUT NAME $ ID AGE PARTY PHONENUM;
   FORMAT PARTY P. PHONENUM PHONE.;
   CARDS;
JOHN  191 18 1 3142345
JOYCE 218 19 2 9872987
HARRY 923 21 4 2343323
JAMES 432 20 3 2344432
;
PROC PRINT;
```

To store additional formats in an existing PDS, use the LIBRARY=SASLIB option in PROC FORMAT as in the example above; the only change is that the SASLIB DD statement now refers to an existing data set into which the formats are written. For example:

```
// EXEC SAS
//SASLIB DD DSN=ACCT.ME.STORE,DISP=(OLD,KEEP)
PROC FORMAT LIBRARY=SASLIB;
   PICTURE STUDNTID OTHER='000-000-000';
```

TSO

There are two methods of creating and storing formats with PROC FORMAT under TSO. If the TSO command facility is available at your installation, you can use either method. (The TSO command facility is available if the TSO statement works.) See the chapter, "SAS Statements Used Anywhere," for a description of the TSO statement.) If the TSO facility is not available, you can only use the method described under **TSO command facility not available**.

TSO command facility available To create temporary formats, simply use a PROC FORMAT statement without the LIBRARY= option. You do not need to allocate any files.

To create permanent formats, allocate a PDS with the fileref SASLIB:

```
ALLOC FI(SASLIB) DA(data.set.name) SPACE(primary) TRACKS
  VOL(volser) DIR(directory)
```

with the appropriate information. (See your computing center staff for the necessary information.)

Use this PROC FORMAT statement:

```
PROC FORMAT LIBRARY=SASLIB;
```

to create permanently stored formats. The format's name becomes the PDS member name. You can create some permanent formats and some temporary formats in the same SAS session by using a PROC FORMAT statement with LIBRARY=SASLIB for the permanent formats and a PROC FORMAT statement without that option to create the temporary formats.

If you create a SAS CLIST that does not contain the SASCP command, the SAS System does not read formats from the library identified by the LIBRARY fileref, even if the CLIST contains an ALLOCATE command for the LIBRARY fileref.

TSO command facility not available To create temporary formats with PROC FORMAT, first allocate a file with the fileref, SASLIB, as follows:

```
ALLOC FI(SASLIB) SPACE(primary secondary)
                 TRACKS DIR(directory)
```

In most cases, a value of 20 for the primary and secondary allocations, and 10 for the directory is sufficient. This file is not permanent because a data set name was not specified when it was allocated.

Use the option LIBRARY=SASLIB in the PROC FORMAT statement.

```
PROC FORMAT LIBRARY=SASLIB;
```

The SAS System stores the formats in a temporary file and retrieves them for use in your SAS session.

To create permanent formats, first allocate a file as follows:

```
ALLOC FI(SASLIB) DA(data.set.name) SPACE(primary secondary)
```

```
TRACKS VOL(volser) DIR(directory)
```

Specify the data set name, primary and secondary allocations, volume serial, and directory space as appropriate (see your computing center staff for this information). This is a permanent file because a data set name was specified when it was allocated.

To store formats, use the LIBRARY=SASLIB option in the PROC FORMAT statement:

```
PROC FORMAT LIBRARY=SASLIB;
```

To use stored formats in another SAS session, allocate the file SASLIB with the appropriate data set name as follows:

```
ALLOC FI(SASLIB) DA(data.set.name) SHR
```

You can also use these formats in OS batch jobs by specifying the SASLIB DD statement in your control language as described earlier.

CMS and VM/PC

For CMS and VM/PC, all user-written formats are written to loose TEXT files, whether they are temporary formats or permanent formats.

If you choose to do so, once you have created formats, you can use the CMS TXTLIB command to bundle the TEXT files containing the formats into a TXTLIB. SAS can use formats that are stored in a TXTLIB but will not write formats directly to a TXTLIB.

To create temporary formats, **do not** specify the LIBRARY= option in the PROC FORMAT statement. When SAS executes a PROC FORMAT statement without the LIBRARY= option, the formats created are temporary and are erased at the end of the SAS session.

To create permanent formats, specify LIBRARY=LIBRARY in the PROC FORMAT statement. The format's name becomes the TEXT file's filename. For example, when this PROC FORMAT step executes

```
PROC FORMAT LIBRARY=LIBRARY;
   VALUE VOTER 1='DEMOCRAT'...;
```

a permanent file is created with filename VOTER and filetype TEXT. This TEXT file is not erased at the end of the SAS session. (Any value can be specified for LIBRARY=, but we recommend LIBRARY.)

(Note: the default disk for TEXT files is the write-access minidisk with the most available space, usually your A-disk. To store a format TEXT file on another disk, issue a FILEDEF that specifies LIBRARY as the DDname and the filemode of your choice. Dummy values can be specified for filename and filetype.)

If after creating permanent formats you want to combine them in a TXTLIB, use the CMS TXTLIB command. Formats moved to a TXTLIB can be accessed by SAS as described below.

To use existing user-written formats, simply name the format in the appropriate SAS statement. When SAS determines that the format is not a standard SAS format, it looks for a TEXT file with a matching filename. If no TEXT file is found, SAS searches all GLOBALed TXTLIBs to find the format.

If you have moved a permanent format to a TXTLIB, you must specify the SASLIB option in the SAS command in order to use the format. For example:

```
SAS (SASLIB filename
```

where filename is the filename of the TXTLIB containing the formats. When the SASLIB option appears, SAS issues a GLOBAL command for the TXTLIB with the specified filename. This makes the TXTLIB available for searching.

VSE

To create temporary formats under VSE, use a PROC FORMAT statement without a LIBRARY= option. For example:

```
PROC FORMAT;
   VALUE AGE 1-19='<20'
             20-39='20-39'
             40-64='40-64
             65-HIGH='65+';
```

PROC FORMAT punches permanent formats as object decks when the DECK option is specified in the PROC FORMAT statement. The object deck for a format is link edited in a separate job step. In addition, an entry in the SAS SASVUMOD table must be added to describe the format. These steps are summarized below:

1. Assign SYSPCH to a temporary disk or tape file, or to a card punch.
2. Use the SYSPCH file produced by PROC FORMAT as input to the MAINT program, which catalogs the object deck(s) in a relocatable library.
3. Link edit the format(s) into a suitable core image library.
4. Place the SSI information for the new format(s) in the SASVUMOD table.

These steps are discussed in more detail below.

Step 1: writing the object deck(s) to SYSPCH When you specify the DECK option in the PROC FORMAT statement, the object deck produced is written to the VSE logical unit SYSPCH. Note that PROC FORMAT ignores the // OPTION DECK JCL statement. SYSPCH can be assigned to disk, tape, a real card punch, a virtual card punch spooled by VSE/POWER, or to any other device supported by DTFDI. For a disk or tape file, use the standard VSE fileref (filename) IJSYSPH. If PROC FORMAT is executed under ICCF, you can use the ICCF punch area. If a temporary SYSPCH file is defined in partition standard labels for use by compilers, you can use that file with PROC FORMAT, provided that it does not overlap the SAS WORK file or any SORT work files used.

Any number of object decks can be written to SYSPCH during a single execution of SAS (limited only by the space available in the file). You can execute PROC FORMAT as many times as necessary during a SAS run.

Step 2: cataloging the object deck(s) Use the IBM MAINT utility to catalog object decks in a relocatable library. The SYSPCH file created by PROC FORMAT should be assigned as SYSIPT to the MAINT utility. Each object deck created by PROC FORMAT is preceded by a MAINT CATALR control card to catalog the deck in the relocatable library. The name of the object module is the name you specified in the VALUE or PICTURE statement that defined the format.

Step 3: link editing the format(s) Use the VSE linkage editor to place the format(s) in a suitable core image library. You can use the core image library for the base SAS product, or you may want to create a separate core image library for your own formats.

The name specified in the linkage editor PHASE statement is the name you subsequently use to refer to the format. You can use any valid format name. Do not use the name of any phase already in any of the SAS product core image libraries. The changed format name must be the name added to the SSI table in SASVUMOD (described below) unless it is already in that table with the correct SSI information.

Step 4: adding SSI information to the SASVUMOD table SAS procedures and DATA steps require SSI information (SSI stands for System Status Information) to

use formats. Under VSE, SAS obtains this information from a table in phase SASVUMOD for user-supplied formats. You must place the appropriate SSI information in the SASVUMOD table to use a new format. Then reassemble and link edit the table.

Updating the SASVUMOD table is described in SAS Technical Report P-125, "Adding User-written Programs to VSE SAS." PROC FORMAT writes the required SSI information on the SAS log. Specify SSI information in SASVUMOD exactly as it is printed on the SAS log.

If you update an existing format, the SSI in SASVUMOD may be correct already. Check that the SSI information printed on the SAS log agrees with that specified in SASVUMOD. If not, update the SASVUMOD entry; then reassemble and link SASVUMOD as described above.

AOS/VS

To create temporary formats, do not specify the LIBRARY= option in the PROC FORMAT statement. The formats are stored in files that are automatically deleted at the end of the SAS job.

To create permanent formats for use in later SAS jobs, follow these steps:

1. Decide on a directory in which to store the formats.
2. Use a LIBNAME statement to associate a library reference name with that directory.
3. In the PROC FORMAT statement use that libref as the value of the LIBRARY= option.

This example stores a numeric and a character format permanently:

```
LIBNAME SAVE ':MY:PERMFMTS';
PROC FORMAT LIBRARY=SAVE;
   VALUE DENSITY 1='URBAN' 2='SUBURBAN';
   VALUE $CITY 'R'='RALEIGH' 'C'='CARY' 'A'='APEX' 'G'='GARNER';
```

To use these formats in another SAS job, follow these steps:

1. Use a LIBNAME statement to assign a libref to the directory containing the formats. The libref does not have to be the same libref you used in the job that created the formats. In addition, if you plan to use formats from more than one directory, assign a libref to each directory.
2. Use that libref (librefs) in a LIBSEARCH statement.
   Note: if two or more directories contain formats of the same name, the SAS System uses the format in the directory whose libref appears earlier in the LIBSEARCH statement; thus, the order of librefs in the LIBSEARCH statement may be important. If the SAS System does not find the format in any of the directories in the search list, it then searches the WORK library (usually your current default directory) for the formats.
3. Use the formats in the appropriate FORMAT statement.
   Note: SAS formats are stored as files in the directory specified in the LIBNAME statement with the following file extension:
   SFC a file containing permanent character formats
   SFN a file containing permanent numeric formats.

This example uses the formats stored in the example above in a new SAS job:

```
LIBNAME GETFMTS ':MY:PERMFMTS';
LIBSEARCH GETFMTS;
PROC FREQ DATA=SURVEY2;
   TABLES ITEM8*ITEM9;
```

```
   FORMAT ITEM8 DENSITY. ITEM9 $CITY.;
```

The following example uses formats stored in two directories:

```
LIBNAME GETFMTS ':MY:PERMFMTS' GETOTHER ':THEIR:FORMATS';
LIBSEARCH GETFMTS GETOTHER;
PROC PRINT DATA=SURVEY3;
   VAR ITEM4 ITEM6 ITEM12;
   FORMAT ITEM4 DENSITY. ITEM6 INCOME.
      ITEM12 $CITY.;
```

Note that to include both librefs (GETFMTS and GETOTHER) in the search list, you must list them in the same LIBSEARCH statement. Specifying

```
LIBSEARCH GETFMTS;
LIBSEARCH GETOTHER;
```

causes the second search list to replace the first one.

PRIMOS

To create temporary formats, do not specify the DDNAME= option in the PROC FORMAT statement. The formats are stored in files that are automatically deleted at the end of the SAS job.

To create permanent formats for use in later SAS jobs, follow these steps:

1. Decide on a directory in which to store the formats.
2. Use a LIBNAME statement to associate a library reference name with that directory.
3. In the PROC FORMAT statement use that libref as the value of the LIBRARY= option.

This example stores a numeric and a character format permanently:

```
LIBNAME SAVE ':MY:PERMFMTS';
PROC FORMAT LIBRARY=SAVE;
   VALUE DENSITY 1='URBAN' 2='SUBURBAN';
   VALUE $CITY 'R'='RALEIGH' 'C'='CARY' 'A'='APEX' 'G'='GARNER';
```

To use these formats in another SAS job, follow these steps:

1. Use a LIBNAME statement to assign a libref to the directory containing the formats. The libref does not have to be the same libref you used in the job that created the formats. In addition, if you plan to use formats from more than one directory, assign a libref to each directory.
2. Use that libref (librefs) in a LIBSEARCH statement.
   Note: if two or more directories contain formats of the same name, the SAS System uses the format in the directory whose libref appears earlier in the LIBSEARCH statement; thus, the order of librefs in the LIBSEARCH statement can be important. If the SAS System does not find the format in any of the directories in the search list, it then searches the WORK library (usually your current default directory) for the formats.
3. Use the formats in the appropriate FORMAT statement.
   Note: SAS formats are stored as files in the directory specified in the LIBNAME statement with the following file extension:
   SFC a file containing permanent character formats
   SFN a file containing permanent numeric formats.

This example uses the formats stored in the example above in a new SAS job:

```
LIBNAME GETFMTS 'MY>PERMFMTS';
LIBSEARCH GETFMTS;
PROC FREQ DATA=SURVEY2;
   TABLES ITEM8*ITEM9;
   FORMAT ITEM8 DENSITY. ITEM9 $CITY.;
```

The following example illustrates using formats stored in two directories:

```
LIBNAME GETFMTS 'MY>PERMFMTS' GETOTHER ':THEIR:FORMATS';
LIBSEARCH GETFMTS GETOTHER;
PROC PRINT DATA=SURVEY3;
   VAR ITEM4 ITEM6 ITEM12;
   FORMAT ITEM4 DENSITY. ITEM6 INCOME.
      ITEM12 $CITY.;
```

Note that to include both librefs (GETFMTS and GETOTHER) in the search list, you must list them in the same LIBSEARCH statement. Specifying

```
LIBSEARCH GETFMTS;
LIBSEARCH GETOTHER;
```

causes the second search list to replace the first one.

VMS

To create temporary formats, do not specify the LIBRARY= option in the PROC FORMAT statement. The formats are stored in files that are automatically deleted at the end of the SAS job.

To create permanent formats for use in later SAS jobs, follow these steps:

1. Decide on a directory in which to store the formats.
2. Use a LIBNAME statement to associate a library reference name with that directory.
3. In the PROC FORMAT statement use that libref as the value of the LIBRARY= option.

This example stores a numeric and a character format permanently:

```
LIBNAME SAVE '[MY.PERMFMTS]';
PROC FORMAT LIBRARY=SAVE;
   VALUE DENSITY 1='URBAN' 2='SUBURBAN';
   VALUE $CITY R='RALEIGH' C='CARY' A='APEX' G='GARNER';
```

To use these formats in another SAS job, follow these steps:

1. Use a LIBNAME statement to assign a libref to the directory containing the formats. The libref does not have to be the same libref you used in the job that created the formats. In addition, if you plan to use formats from more than one directory, assign a libref to each directory.
2. Use that libref (librefs) in a LIBSEARCH statement.
   Note: if two or more directories contain formats of the same name, the SAS System uses the format in the directory whose libref appears earlier in the LIBSEARCH statement; thus, the order of librefs in the LIBSEARCH statement can be important. If the SAS System does not find the format in any of the directories in the search list, it then searches the WORK library (usually your current default directory) for the formats.
3. Use the formats in the appropriate FORMAT statement.
   Note: SAS formats are stored as files in the directory specified in the LIBNAME statement with the following file type:

SFC a file containing permanent character formats
SFN a file containing permanent numeric formats.

This example uses the formats stored in the example above in a new SAS job:

```
LIBNAME GETFMTS '[MY:PERMFMTS]';
LIBSEARCH GETFMTS;
PROC FREQ DATA=SURVEY2;
   TABLES ITEM8*ITEM9;
   FORMAT ITEM8 DENSITY. ITEM9 $CITY.;
```

The following example illustrates using formats stored in two directories:

```
LIBNAME GETFMTS '[MY:PERMFMTS]' GETOTHER '[THEIR:FORMATS]';
LIBSEARCH GETFMTS GETOTHER;
PROC PRINT DATA=SURVEY3;
   VAR ITEM4 ITEM6 ITEM12;
   FORMAT ITEM4 DENSITY. ITEM6 INCOME.
      ITEM12 $CITY.;
```

Note that to include both librefs GETFMTS and GETOTHER in the search list, you must list them in the same LIBSEARCH statement. Specifying

```
LIBSEARCH GETFMTS;
LIBSEARCH GETOTHER;
```

causes the second search list to replace the first one.

NOTES

1. **AOS/VS, PRIMOS, and VMS**: In these environments, user-written formats are stored as SAS files; therefore, you must use a libref when specifying LIBRARY=. The type of the SAS file is FORMATC for character formats and FORMATN for numeric formats.

   CMS, OS, VM/PC, and VSE: in these environments, user-written formats are stored in external files; therefore, you must use a fileref when specifying LIBRARY=.

2. **VSE**: An additional restriction is that the name must not be the name of any SAS System module. If a format name duplicates a SAS System module name, the program terminates abnormally.

3. **CMS, OS, VM/PC, and VSE**: All labels and all character values must be in quotes, unless the SAS System option TEXT82 is in effect.

4. See note 3.

5. **VSE**: Using a reserved SAS word to name the format is a fatal error.

Chapter 36

The FORMS Procedure

Operating systems: All

ABSTRACT

The FORMS procedure can produce labels for envelopes, mailing labels, external tape labels, file cards, and any other printer forms that have a regular pattern.

INTRODUCTION

For each observation in the input SAS data set, PROC FORMS prints data in a rectangular block called a **form unit**. For example, a mailing label is a form unit.

The results can be routed to a specified external file or printed on the SAS print file. For external files, carriage controls can be used or suppressed.[1]

Form layout The size and spacing of the forms are controlled by options as illustrated in **Figure 36.1**. The keyword options for the PROC FORMS statement are shown with their abbreviations in upper case.

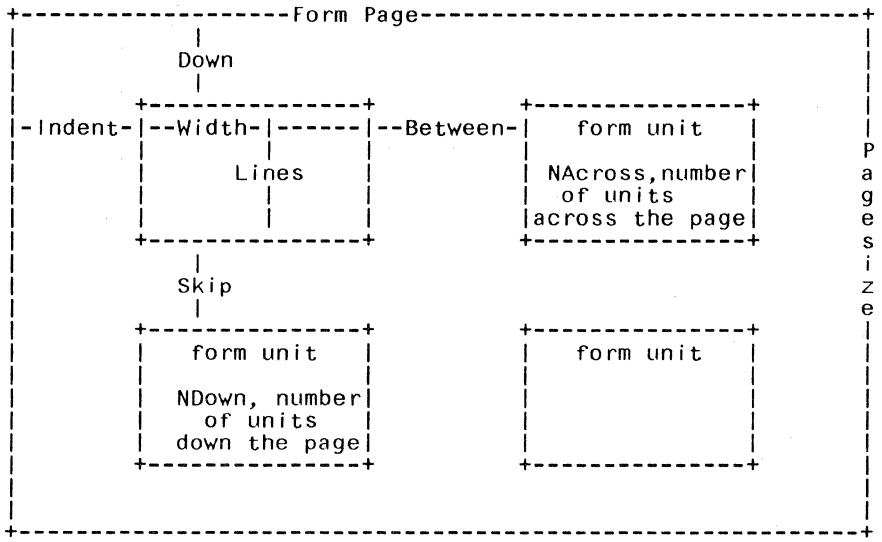


Figure 36.1 Sample PROC FORMS Placement

The values specified in LINE statements are formatted into a form unit that is WIDTH columns wide, and LINES lines long. Values that do not fit into WIDTH columns are truncated. NACROSS form units are printed across the page, with BETWEEN columns between adjacent form units. The forms are indented INDENT spaces from the left margin. SKIP blank lines are printed between form units down the page.

PROC FORMS has two modes of operation: continuous mode and page mode. Page-mode operation is used if NDOWN= or PAGESIZE= options are specified or if the SAS print file is used rather than a DDNAME= file.

Continuous-mode operation For continuous-mode operation, the procedure first skips DOWN= spaces, then alternately prints LINES= print lines and SKIP= blank lines until all the data are printed. No carriage control characters are used to skip to the top of the form. Continuous-mode operation is suitable for most forms unless the distance between form units changes between pages. [2]

Page-mode operation First, PROC FORMS goes to the start of a new page and then prints DOWN blank lines (see **Carriage Control** below). Then NDOWN sets of form units are printed down the page. If the NDOWN= option is not specified, the number of sets of form units will be set to the maximum allowed by the PAGESIZE= value specified or assumed.

If the SAS print file is used (which occurs if DDNAME= is not specified), all the page calculations are done with respect to the lines remaining on the page after the title lines are printed.

Carriage control When you route the forms to an external file specified by the DDNAME= option, you must choose whether to use carriage control characters.[3] A carriage control is an extra character prefixed to each record that controls the printer for line spacing and top-of-form ejection. If you want PROC FORMS to use carriage controls, you must specify the CC option in the PROC FORMS statement and an appropriate specification for the record format of the print file.[4] If you do not use carriage controls, PROC FORMS prints blank lines to align form units correctly on the page.

If you are using page-mode printing with the CC option, PROC FORMS uses the top-of-form carriage control to start each page. For page-mode operation without CC, PAGESIZE= must be correct since FORMS uses it to calculate the number of blank lines to print at the bottom of a page to get to the next page.

Multiple Copies

Three features in PROC FORMS provide for multiple copies of forms. If you want the whole list repeated as a unit, use the SETS= option. If you want each observation repeated in adjacent form units, use the COPIES= option. If you want a variable number of copies depending on a variable in a data set, use the FREQ statement. These options can be used separately or together.

SPECIFICATIONS

The statements used to control PROC FORMS are

PROC FORMS *options*;
LINE *linenumber variables / options*;
FREQ *variable*;
BY *variables*;

PROC FORMS Statement

PROC FORMS *options*;

The options listed below can appear in the PROC FORMS statement:

DATA=*SASdataset*
: names the input data set for PROC FORMS. If DATA= is omitted, the most recently created SAS data set is used.

DDNAME=*fileref* DD=*fileref*
: specifies an external file where PROC FORMS writes the forms. If not specified, PROC FORMS uses the SAS print file.[5] Consult the section **Using External Files** for details.

Options to control form placement

WIDTH=*number*
W=*number*
: specifies the number of print positions across the form unit. The minimum is 1, the maximum is 255, and the default is 40.

LINES=*number*
L=*number*
: specifies the number of lines on a form unit. The minimum is 1, the maximum is 255, and the default is the maximum LINE statement number.

DOWN=*number*
D=*number*
: specifies the number of lines to skip on a page before printing the first line. The minimum is 0, the maximum is 200, and the default is 0. [6]

SKIP=*number*
S=*number*
: specifies the number of lines to skip between form units. The minimum is 0, the maximum is 200, and the default is 1.

NACROSS=*number*
NA=*number*
: specifies the number of form units across the page. The minimum is 1, the maximum is 200, and the default is 1.

BETWEEN=*number*
B=*number*
: specifies the number of print positions between form units. The minimum is 0, the maximum is 200, and the default is 1.

INDENT=*number*
I=*number*
: specifies the number of positions to indent the first form unit across the page. The minimum is 0, the maximum is 200, and the default is 0.

Page layout options

NDOWN=*number*
ND=*number*
: specifies the number of form units printed down the page. This option triggers page-mode operation. The default is

```
FLOOR((PAGESIZE-DOWN+SKIP)/(LINES+SKIP))
```

PAGESIZE=*number* P=*number*
: specifies the number of lines on a form page. This information is only needed for page-mode operation if CC is not specified. The minimum is DOWN+LINES, the maximum is 255, and the default is 66 if DDNAME= is specified; otherwise the number of lines is inferred from SAS print file characteristics and titling information. This option triggers page-mode operation.

Other options

CC
: requests that the procedure add carriage control characters to the beginning of each print line. This option is only relevant if the DDNAME= option is specified.

COPIES=*number*
C=*number*
: specifies that *number* copies be produced for each observation in the input data set. The copies are produced together rather than in separate sets (as in the SETS= option). In adjacent form units, the number of copies produced is the product of the value of the COPIES= option multiplied by the FREQ variable.

SETS=*number*
: specifies that *number* multiple copies of the entire list be printed. For page-mode operation, PROC FORMS skips to a new page to start each set. This option is different from COPIES=, which prints the multiples in adjacent positions.

ALIGN=*number*
: controls the number of XXXX form units printed so that the printer can be aligned. The default is 8 if DDNAME= is specified; otherwise it is 0.

LINE Statement

LINE *linenumber variables / options;*

LINE statements come directly after the PROC FORMS statement and specify the information to be printed on each line of the form unit.

The *linenumber* value gives the number of the line, and it must be an integer from 1 to the value of the LINES= option specified in the PROC FORMS statement. A LINE statement need not be given for a blank line.

Values of the *variables* specified in the LINE statement are printed on the specified line of the form unit. Up to 100 variables may be given. One blank space is inserted between each value. If the length of the values is longer than the WIDTH= value specified in the PROC FORMS statement, the field is truncated to the WIDTH= value specified. The PACK option can squeeze out extra blanks to get more fields onto a small form unit. However, if you must squeeze fields onto a form unit, it is better to use a FORMAT statement to truncate each field individually, rather than risk losing an entire field. For example,

```
FORMAT CITY $20. STATE $2.;
```

reduces the defined width of variables CITY and STATE to 20 and 2 columns, respectively.

The options below can appear in LINE statements after a slash (/).

INDENT=*number*
I=*number*
: specifies that the line is to be indented by the value of *number*. This indent is done within the form unit (in contrast to the INDENT= option for the PROC FORMS statement).

PACK
P
: removes extra blanks from the line so that all the values are separated by one space. For example, if a variable NAME has a length of 20 and the value of NAME is 'SMITH', the PACK option removes the extra 15 blanks.

LASTNAME
L
: looks for commas in a character variable containing a name and rotates the words after the comma if it finds one. For example, a variable NAME might have the value 'SMITH, JAMES P.'. If the LASTNAME option is given, the value is printed 'JAMES P. SMITH'.

REMOVE
R
: deletes the entire line if the values of the variables are all blanks or missing values.

FREQ Statement

FREQ *variable;*

When a FREQ statement is included, the value of the frequency variable determines the number of form units printed for each observation.

The variable given in the FREQ statement must be numeric. If the value is not an integer, the integer portion of the value determines the number of form units printed. If the value is less than 1 or is missing, one form unit is printed for each observation.

The actual number of copies produced in adjacent form units is the product of the FREQ variable multiplied by the value of the COPIES= option. For example, if the FREQ variable is 3 and the COPIES value is 2, then 6 copies are printed.

BY Statement

BY *variables*;

A BY statement can be used with PROC FORMS to force new pages to be started (in page-mode operation) when the BY values change. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

DETAILS

Missing Values

PROC FORMS prints missing values as blanks.

Using External Files and Special Forms

The DDNAME= option directs PROC FORMS to route the output to an external file.[7]

You must also provide for the file with suitable control statements because you may need to request special forms to be mounted in your printer.[8] Your installation's rules determine how special forms are handled. Check with your computer center staff for specific details.[9]

At many installations, the operating system automatically skips to a new page after printing a given number of lines. Check with your computer center staff to disable this skip for continuous forms, such as labels. [10]

Compatibility Features

PROC FORMS is similar to the earlier FORMS procedure in the supplemental library. The following options are still maintained to ensure compatibility: NSKIP and NS for SKIP, NBETWEEN and NB for BETWEEN, NCOPIES and N for COPIES.

The former PROC FORMS options BURST, TRUNCATE, NOALIGN, and EIGHTLPI have been replaced by more efficient options. The CODE= option is still supported. The DOWN option has been slightly modified.

EXAMPLE

Printing a List of Names and Addresses Using PROC FORMS

The job below prints a listing of a set of names and addresses.

```
DATA MAILLIST;
    INPUT NAME &$20. STREET &$20. CITY &$15. STATE :$2. ZIP :$5.;
    CARDS;
JOHN SMITH          202 MAIN ST.         ANYTOWN      IN  22432
JOE WILLIAMS        101 BROADWAY         GOTHAM       NY  32334
ALICE GORDON        55 HAZEL WAY         ATLANTA      GA  68549
```

```
GEORGE HAYES        2132 GOODVIEW        MUNCIE       IN  54345
PENELOPE PETERS     543 PEANUT CIRCLE    COLUMBIA     MD  53455
HARRISON HARRISON   898 7TH ST           BIG ROCK     AK  98733
BILL SMITH          123 STATE            ROCKFORD     IL  81522
WILL RED            544 HILLSIDE AV      ROCQUEFORT   KY  98783
ANN ANDERSON        5456 PLEASANT BLVD   SEATTLE      WA  49747
ARISTOTLE JOHNSON   234 PEACE            PHOENIX      AZ  77898
GARRY MILLER        7 GRAY DRIVE         ST. LOUIS    MO  26578
WINSTON MARTIN      34 RED ROAD          NEWTON       MA  34678
ALISON TOWSON       44 MARS HILL DR.     REDWOOD      CA  23432
MATHEW MACKEM       78 HIGHVIEW CT.      VIRGIL       VA  87373
;
PROC SORT;
    BY ZIP;
PROC FORMS WIDTH=25 LINES=3 DOWN=2 SKIP=2 NACROSS=4 BETWEEN=1;
    LINE 1 NAME;
    LINE 2 STREET;
    LINE 3 CITY STATE ZIP/PACK;
```

Output 36.1 Mailing List Produced by PROC FORMS

```
JOHN SMITH              ALISON TOWSON           GARRY MILLER            JOE WILLIAMS
202 MAIN ST.            44 MARS HILL DR.        7 GRAY DRIVE            101 BROADWAY
ANYTOWN IN 22432        REDWOOD CA 23432        ST. LOUIS MO 26578      GOTHAM NY 32334

WINSTON MARTIN          ANN ANDERSON            PENELOPE PETERS         GEORGE HAYES
34 RED ROAD             5456 PLEASANT BLVD      543 PEANUT CIRCLE       2132 GOODVIEW
NEWTON MA 34678         SEATTLE WA 49747        COLUMBIA MD 53455       MUNCIE IN 54345

ALICE GORDON            ARISTOTLE JOHNSON       BILL SMITH              MATHEW MACKEM
55 HAZEL WAY            234 PEACE               123 STATE               78 HIGHVIEW CT.
ATLANTA GA 68549        PHOENIX AZ 77898        ROCKFORD IL 81522       VIRGIL VA 87373

HARRISON HARRISON       WILL RED
898 7TH ST              544 HILLSIDE AV
BIG ROCK AK 98733       ROCQUEFORT KY 98783
```

NOTES

1. **VSE**: The only external file that can be used is SYSLST, and carriage controls cannot be used.
2. **VSE**: Continuous mode operation is always used if you specify the DDNAME= option.
3. **VSE**: This applies to all operating systems except VSE.
4. **OS**: OS batch jobs, specify the print file attributes:

```
DCB=(RECFM=VBA,LRECL=137,BLKSIZE=141)
```

5. **CMS**: This corresponds to the DDname of a FILEDEF.

   OS: Under OS batch, the fileref corresponds to the DDname in the JCL DD statement, and under TSO this corresponds to an ALLOCATE command filename.

VSE: Using DDNAME= always routes the output to SYSLST and sets continuous mode. The actual DDname specified is ignored. For **AOS/VS, PRIMOS,** and **VMS** the DDname is the fileref defined on the FILENAME statement.

6. **VMS**: This option skips an additional line. This results from a redefinition of the title lines to include an additional blank line after the last printed title line.
7. **VSE**: The output is always routed to SYSLST if you specify DDNAME=.
8. For example, suppose PROC FORMS prints mailing labels that have a special-forms number at your installation of 6666. The following JCL and SAS statements put the FORMS output on special form 6666:

```
//  EXEC SAS
//LABELS DD SYSOUT=(A,,6666)
//DCB=(RECFM=VBA,LRECL=137,BLKSIZE=141)
.
. SAS Statements
.
PROC FORMS DDNAME=LABELS...;
```

9. **AOS/VS**: QPRINT/FORM=installation defined codes.
   PRIMOS: SPOOL - FORM installation defined type.
   VMS: PRINT/FORM=installation defined codes.
10. **OS**: One convention used in JES2 is

```
/*JOBPARM LINECNT=0
```

Chapter 37

The FREQ Procedure

Operating systems: All

ABSTRACT

The FREQ procedure produces one-way to *n*-way frequency and crosstabulation tables. For two-way tables, PROC FREQ computes tests and measures of association. For *n*-way tables, PROC FREQ does stratified analysis, computing statistics within as well as across strata. Frequencies can also be output to a SAS data set.

INTRODUCTION

Frequency tables show the distribution of variable values. For example, if a variable A has six possible values, a frequency table for A shows how many observations in the data set have the first value of A, how many have the second value, and so on.

Crosstabulation tables show combined frequency distributions for two or more variables. For example, a crosstabulation table for the variables SEX and EMPLOY

shows the number of working females, the number of non-working females, the number of working males, and the number of non-working males.

One-Way Frequency Tables

If you want a one-way frequency table for a variable, simply name the variable in a TABLES statement. For example, the statements

```
PROC FREQ;
   TABLES A;
```

produce a one-way frequency table giving the values of A and the frequency of each value.

Two-Way Crosstabulation Tables

If you want a crosstabulation table for two variables, give their names separated by an asterisk (*). Values of the first variable form the rows of the table, and values of the second variable form the columns. For example, the statements

```
PROC FREQ;
   TABLES A*B;
```

produce a crosstabulation table with values of A down the side and values of B across the top.

N-Way Crosstabulation Tables

If you want a three-way (or *n*-way) crosstabulation table, give the three (or *n*) variable names separated by asterisks in the TABLES statement. Values of the last variable form the columns of a contingency table; values of the next-to-last variable form the rows. Each level (or combination of levels) of the other variables form one stratum, and a separate contingency table is produced for each stratum. For example, the statements

```
PROC FREQ;
   TABLES A*B*C*D / CMH;
```

produce *k* tables, where *k* is the number of different combinations of values for the variables A and B. Each table has the values of C down the side and the values of D across the top.

Note: multi-way tables can generate a great deal of printed output. For example, if the variables A, B, C, D, and E each have ten levels, five-way tables of A*B*C*D*E could generate 4000 or more pages of output.

PROC FREQ Contrasted with Other SAS Procedures

Many other procedures in SAS can collect frequency counts. PROC FREQ is distinguished by its ability to compute chi-square tests and measures of association for two-way and *n*-way tables. Other procedures to consider for counting are the following: TABULATE for more general table layouts, SUMMARY for output data sets, and CHART for bar charts and other graphical representations. PROC CATMOD can be used for general linear model analysis of categorical data.

SPECIFICATIONS

The statements available in PROC FREQ are

PROC FREQ *options*;

TABLES *requests / options*;
WEIGHT *variable*;
BY *variables*;

PROC FREQ Statement

PROC FREQ *options*;

The options that can be used in the PROC FREQ statement are as follows:

DATA=*SASdataset*
: specifies the data set to be used by PROC FREQ. If the DATA= option is omitted, FREQ uses the most recently created data set.

ORDER=FREQ
ORDER=DATA
ORDER=INTERNAL
ORDER=FORMATTED
: specifies the order in which the variable levels are to be reported. If ORDER=FREQ, levels are ordered by descending frequency count so that the levels with the largest frequencies come first. If ORDER=DATA, levels are put in the order in which they first occur in the input data. If ORDER=INTERNAL, then the levels are ordered by the internal value. If ORDER=FORMATTED, levels are ordered by the external formatted value. If you omit ORDER= or give an unrecognized value, PROC FREQ orders by the internal value.

FORMCHAR(1,2,7)='*string*'
: defines the characters to be used for constructing the outlines and dividers for the cells of contingency tables. The string should be three characters long. The characters are used to denote (1) vertical divider, (2) horizontal divider, and (3) vertical-horizontal intersection. Any character or hexadecimal string can be used to customize table appearance. Specifying FORMCHAR(1,2,7)=' ' (3 blanks) produces tables with no outlines or dividers. If you do not specify the FORMCHAR= option, FREQ uses the default FORMCHAR(1,2,7)='|-+'. See the CALENDAR and TABULATE procedures for further information.

TABLES Statement

TABLES *requests / options*;

For each frequency or crosstabulation table that you want, put a table request in the TABLES statement.

requests
: are composed of one or more variable names joined by asterisks (*). A one-way frequency is generated by a single name. Two-way crosstabulations are generated by two variables joined with an asterisk. Any number of variables can be joined for a multi-way table. A grouping syntax is also available to make the specifications of many tables easier. Several variables can be put in parentheses and joined to other effects.

For example,

TABLES A*(B C);	is equivalent to	TABLES A*B A*C;
TABLES (A B)*(C D);	is equivalent to	TABLES A*C A*D B*C B*D;

TABLES (A B C)*D;	is equivalent to	TABLES A*D B*D C*D;
TABLES A--C;	is equivalent to	TABLES A B C;
TABLES (A--C)*D;	is equivalent to	TABLES A*D B*D C*D;
TABLES A--C*D;	is illegal.	

Any number of requests can be given in one TABLES statement, and any number of TABLES statements can be included in one execution of PROC FREQ. If there is no TABLES statement, FREQ does one-way frequencies for all of the variables in the data set.

If you request a one-way table for a variable and do not specify any options, FREQ produces frequencies, cumulative frequencies, percentages of the total frequency, and cumulative percentages for each level of the variable.

If you request a two-way table and do not specify any options, FREQ produces crosstabulation tables that include cell frequencies, cell percentages of the total frequency, cell percentages of row frequencies, and cell percentages of column frequencies. Missing levels of each variable are excluded from the table, but the total frequency of missing subjects is printed below each table.

The options below can be used in the TABLES statement after a slash (/):

General options

MISSING
: requests FREQ to interpret missing values as nonmissing and to include them in calculations of percentages and other statistics.

LIST
: prints two-way to *n*-way tables in a list format rather than as crosstabulation tables. The LIST option cannot be used when statistical tests or measures of association are requested. Expected cell frequencies are not printed when LIST is specified, even if EXPECTED is specified.

OUT=*SASdataset*
: sets up an output SAS data set containing variable values and frequency counts. If more than one table request appears in the TABLES statement, the contents of the data set correspond to the last table request in the TABLES statement. For details on the output data set created by PROC FREQ, see **Output Data Set** below. If you want to create a permanent SAS data set, you must specify a two-level name. See "SAS Files" for more information on permanent SAS data sets.

Option to request statistical analysis

CHISQ
: requests a chi-square (χ^2) test of homogeneity or independence for each stratum, together with measures of association based on chi-square. The tests include Pearson chi-square, likelihood ratio chi-square, and Mantel-Haenszel chi-square. The measures include the phi coefficient, the contingency coefficient, and Cramer's V. For 2 by 2 tables, Fisher's Exact Test is also included.

Additional statistical options available in the FREQ procedure are fully documented in *SAS User's Guide: Statistics.*

Options to request additional table information

EXPECTED
: requests that the expected cell frequencies under the hypothesis of independence (or homogeneity) be printed.

DEVIATION
: requests that, for each cell, FREQ print the deviation of the cell frequency from the expected value.

CELLCHI2
: requests that FREQ print each cell's contribution to the total χ^2 statistic. This is computed as *(frequency−expected)**2/expected.*

CUMCOL
: requests that cumulative column percentages be printed in the cells.

MISSPRINT
: asks FREQ to print missing value frequencies for two-way to *n*-way tables, even though the frequencies will not be used in the calculation of statistics.

SPARSE
: causes the procedure to write out or print information about all possible combinations of levels of the variables in the table request, even when some combinations of levels do not occur in the data. This option affects printouts under the LIST option and output data sets.

Options to suppress printing

NOFREQ
: suppresses printing of the cell frequencies for a crosstabulation.

NOPERCENT
: suppresses printing of the cell percentages for a crosstabulation.

NOROW
: suppresses printing of the row percentages in cells of a crosstabulation.

NOCOL
: suppresses printing of the column percentages in cells of a crosstabulation.

NOCUM
: suppresses printing of the cumulative frequencies and cumulative percentages for one-way frequencies and for frequencies in list format.

NOPRINT
: suppresses printing of the tables, but allows printing of the statistics specified by CHISQ, MEASURES, CMH, and ALL. (Complete descriptions of the CHISQ, MEASURES, CMH, and ALL options are given in the chapter "The FREQ Procedure" in *SAS User's Guide: Statistics.*)

WEIGHT Statement

WEIGHT *variable*;

Normally, each observation contributes a value of 1 to the frequency counts. (In other words, each observation represents one subject.) However, when a

WEIGHT statement appears, each observation contributes the weighting variable's value for that observation. (For example, a weight of 3 means that the observation represents 3 subjects.) The values must be nonnegative, but they do not have to be integers. FREQ uses double precision floating point arithmetic to accumulate the counts or weights. Values are summed and then printed with decimal places, if appropriate.

Only one WEIGHT statement can be used, and that statement applies to counts collected for all tables.

For example, suppose a data set contains variables RACE, SEX, and HRSWORK. The statements

```
PROC FREQ;
   TABLES RACE*SEX;
```

produce a table showing how many nonwhite females, nonwhite males, white females, and white males are present. The statements

```
PROC FREQ;
   TABLES RACE*SEX;
   WEIGHT HRSWORK;
```

produce a table showing the number of hours worked by nonwhite females, by nonwhite males, and so on.

BY Statement

BY *variables*;

A BY statement can be used with PROC FREQ to obtain separate analyses for the groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

DETAILS

Missing Values

For one-way frequency tables, the missing value frequencies appear in the tables. For two-way to *n*-way tables, the missing value frequencies do not appear, but the total frequency of missing subjects is given below each table. In all cases, the statistics do not include missing values.

Missing value frequencies can be printed by specifying the MISSPRINT option in the TABLES statement; they can be included in the computation of statistics by specifying the MISSING option.

Limitations

Any number of TABLES statements can be included after the PROC FREQ statement. Since FREQ builds all the tables requested in all TABLES statements in one pass of the data, there is essentially no loss of efficiency from using multiple TABLES statements.

A TABLES statement can contain any number of table requests, and each request can include any number of variables. The maximum number of levels allowed for any one variable is 32,000. If you have a variable with more than

32,000 levels, use PROC SUMMARY and PROC PRINT or reduce the number of levels by using the FORMAT statement.

FREQ stores each combination of values in memory. When FREQ is compiling and developing multi-way tables or when some variables have many levels, you may run out of main storage. If increasing the region size is impractical, use PROC SORT to sort the data set by one or more of the variables and then use PROC FREQ with a BY statement that includes the sorted variables.

The FREQ procedure handles both internal and formatted values up to length 16 on both the printout and the output data set. Longer data values are truncated to sixteen characters, and a warning message is printed on the SAS log.

Frequency values with more than seven significant digits may be printed in scientific notation (E format), in which case only the first few significant digits of the mantissa are printed. If you need more significant digits than FREQ prints, you can specify an output data set with the OUT= option. Then use

```
PROC PRINT DATA=FREQDATA;
   FORMAT COUNT BEST32.;
```

where FREQDATA is the OUT= data set produced by PROC FREQ.

The variable COUNT, containing the frequency values, is then printed with additional significant digits.

Output Data Set

The new data set produced by PROC FREQ contains one observation for each combination of the variable values in the table request. Each observation contains these variables plus two new variables, COUNT and PERCENT, which give respectively the frequency and cell percentage for the combination of variable values.

For example, consider the statements

```
PROC FREQ;
   TABLES A A*B / OUT=D;
```

The output data set D corresponds to the rightmost table request, A*B. If A has two values (1 and 2) and B has three values (1, 2, and 3), the output data set D can have up to six observations, one for each combination of the A and B values. In observation 1, A=1 and B=1; in observation 2, A=1 and B=2; and so on. The data set also contains the variables COUNT and PERCENT. COUNT's value in each observation is the number of subjects that have the given combination of A and B values; PERCENT's value is the percent of the total number of subjects having that A and B combination.

When FREQ collects different class values into the same formatted level, it saves the smallest internal value to output in the output data set.

Computer Resources

For each variable, PROC FREQ stores all of the levels in memory, requiring 56 bytes for each level. If FREQ runs out of memory, it stops collecting levels on the variable with the most levels and returns the memory so that counting can continue. The procedure then builds the tables that do not contain the disabled variables.

For two-way and *n*-way tables, FREQ uses a utility file to store frequencies when the number of nonzero cells exceeds 63.

Grouping with Formats

When you use PROC FREQ, remember that FREQ groups the variables according to their formatted values. If you assign a format to a variable with a FORMAT state-

ment, the variable's values are formatted for printing before FREQ divides the observations into groups for the frequency counts.

For example, say a variable X has the values 1.3, 1.7, and 2.0, among others. Each of these values appears as a level in the frequency table. If you want each value rounded to a single digit, you include the statement

```
FORMAT X 1.;
```

after the PROC FREQ statement. The frequency table levels are then 1 and 2.

Formatted character variables are treated in the same way: the formatted values are used to divide the observations into groups. For character variables, formatted or not, only the first sixteen characters are used to determine the groups.

You can also use the FORMAT statement to assign formats created by PROC FORMAT to variables. Formats created by PROC FORMAT can serve two purposes: they can define the levels, and they can label the levels. You can use the same data with different formats to collect counts on different partitions of the class values. For an example, see "PROC Step Applications."

In frequency tables, values of both character and numeric variables appear in ascending order by the original (unformatted) values unless you specify otherwise with the ORDER= option.

Printed Output

For a one-way table showing the frequency distribution of a single variable, FREQ prints these items:

1. the name of the variable and its values
2. FREQUENCY counts giving the number of subjects that have each value
3. CUMULATIVE FREQUENCY counts, giving the sum of the frequency counts of that value and all other values listed above it in the table (the total number of nonmissing subjects is the last cumulative frequency)
4. percentages, labeled PERCENT, giving the percent of the total number of subjects represented by that value
5. CUMULATIVE PERCENT values, giving the percent of the total number of subjects represented by that value and all others previously listed in the table.

Two-way tables can be printed either as crosstabulation tables (the default) or as lists (when the LIST option is specified). Each cell of a crosstabulation table contains items 6 through 12:

6. FREQUENCY counts, giving the number of subjects that have the indicated values of the two variables
7. PERCENT, the percentage of the total frequency count represented by that cell
8. ROW PCT, or the row percentage, the percent of the total frequency count for that row represented by the cell
9. COL PCT, or column percent, the percent of the total frequency count for that column represented by the cell
10. if the EXPECTED option is specified, the expected cell frequency under the hypothesis of independence
11. if the DEVIATION option is specified, the deviation of the cell frequency from the expected value
12. if the CELLCHI2 option is specified, the cell's contribution to the total chi-square statistic.
13. if two contingency tables can fit on a page, one table above the other, then the tables are printed in that manner. Similarly, a table and its

corresponding statistics are printed on the same page, provided that they fit.

EXAMPLE

Census Data Study: Example 1

Data for the following example are selected characteristics of U.S. cities with populations over 50,000 according to the *County and City Data Book, 1983*. The first TABLES statement requests univariate frequencies on all the variables. The second TABLES statement requests a two-way frequency table. Other TABLES statements illustrate different request forms and options of PROC FREQ. The FORMAT statement associates a format that has been previously defined using PROC FORMAT with each variable.

Another example in "PROC Step Applications," shows the DATA step that reads these census data into a SAS data set, the use of PROC UNIVARIATE to determine cutoff points for categories, and PROC FORMAT to define the formats for the variables.

These statements produce the output shown below:

```
DATA CITIES;
more SAS statements
;
PROC FREQ DATA=CITIES
   FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   TABLES DIVISION JANGRP JULYGRP POPGRP REGION INCOMGRP
      CRIMEGRP RAINGRP OVR65GRP HOUSGRP ELECGRP;
   TABLES POPGRP*CITYGOVT;
   TABLES REGION*OVR65GRP / NOCOL NOPERCENT;
   TABLES JANGRP*JULYGRP*ELECGRP / LIST;
   TABLES (POPGRP INCOMGRP)*CRIMEGRP / NOCOL NOPERCENT
      EXPECTED DEVIATION CELLCHI2 CHISQ;
   TABLES CITYGOVT*CRIMEGRP / ALL NOCOL NOPERCENT;
   FORMAT POPGRP PC. OVR65GRP OC. INCOMGRP IC. HOUSGRP HC.
      ELECGRP EC. CITYGOVT GC. CRIMEGRP CC. JANGRP JAN.
      JULYGRP JUL. RAINGRP RC.;
   TITLE 'CENSUS DATA: OUTPUT FROM FREQ PROCEDURE';
```

Output 37.4 Census Data: Output from PROC FREQ

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 1

❶ DIVISION	❷ FREQUENCY	❹ PERCENT	❸ CUMULATIVE FREQUENCY	❺ CUMULATIVE PERCENT
EAST NORTH CENTR	83	19.9	83	19.9
EAST SOUTH CENTR	14	3.3	97	23.2
MIDDLE ATLANTIC	37	8.9	134	32.1
MOUNTAIN	28	6.7	162	38.8
NEW ENGLAND	37	8.9	199	47.6
PACIFIC	90	21.5	289	69.1
SOUTH ATLANTIC	49	11.7	338	80.9
WEST NORTH CENTR	29	6.9	367	87.8
WEST SOUTH CENTR	51	12.2	418	100.0

(continued on next page)

(continued from previous page)

MEAN JANUARY TEMPERATURE (DEGREES F)

JANGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 32 DEGREES	208	49.8	208	49.8
32 DEGREES OR HI	210	50.2	418	100.0

MEAN JULY TEMPERATURE (DEGREES F)

JULYGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 75 DEGREES	241	57.7	241	57.7
75 DEGREES OR HI	177	42.3	418	100.0

1980 POPULATION ESTIMATE

POPGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
50,001-75,000	162	38.8	162	38.8
75,001-100,000	87	20.8	249	59.6
OVER 100,000	169	40.4	418	100.0

REGION	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
NORTH CENTRAL	112	26.8	112	26.8
NORTHEAST	74	17.7	186	44.5
SOUTH	114	27.3	300	71.8
WEST	118	28.2	418	100.0

PER CAPITA MONEY INCOME (1979)

INCOMGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
AVERAGE OR ABOVE	243	58.1	243	58.1
BELOW AVERAGE	175	41.9	418	100.0

CRIME RATE/100,000 POP (1981)

CRIMEGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
6000 OR LESS	109	26.1	109	26.1
6001-7500	102	24.4	211	50.5
7501-9500	107	25.6	318	76.1
OVER 9500	100	23.9	418	100.0

ANNUAL PRECIPITATION (INCHES)

RAINGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LESS THAN 19 INC	99	23.7	99	23.7
19-44 INCHES	228	54.5	327	78.2
OVER 44 INCHES	91	21.8	418	100.0

PERSONS 65 & OVER (1980)

OVR65GRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
7000 OR LESS	111	26.6	111	26.6
7001-17000	209	50.0	320	76.6
OVER 17000	98	23.4	418	100.0

(continued on next page)

(continued from previous page)

HOUSING UNITS (1980)

HOUSGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LOW	103	24.6	103	24.6
MEDIUM	212	50.7	315	75.4
HIGH	103	24.6	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 3

TYPICAL RESID. ELECTRIC BILL/MO. (1982)

ELECGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
LOW	110	26.3	110	26.3
MEDIUM	241	57.7	351	84.0
HIGH	67	16.0	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 4

TABLE OF POPGRP BY CITYGOVT

POPGRP(1980 POPULATION ESTIMATE)
CITYGOVT(FORM OF CITY GOVERNMENT (1982))

6 FREQUENCY 7 PERCENT 8 ROW PCT 9 COL PCT	MAYOR COUNCIL	COUNCIL MANAGER	COMMISSION	TOTAL
50,001-75,000	6 55 7 13.16 8 33.95 9 33.74	104 24.88 64.20 43.33	3 0.72 1.85 20.00	162 38.76
75,001-100,000	33 7.89 37.93 20.25	50 11.96 57.47 20.83	4 0.96 4.60 26.67	87 20.81
OVER 100,000	75 17.94 44.38 46.01	86 20.57 50.89 35.83	8 1.91 4.73 53.33	169 40.43
TOTAL	163 39.00	240 57.42	15 3.59	418 100.00

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 5

TABLE OF REGION BY OVR65GRP

REGION OVR65GRP(PERSONS 65 & OVER (1980))

FREQUENCY ROW PCT	7000 OR LESS	7001-17000	OVER 17000	TOTAL
NORTH CENTRAL	33 29.46	55 49.11	24 21.43	112
NORTHEAST	7 9.46	53 71.62	14 18.92	74
SOUTH	28 24.56	48 42.11	38 33.33	114
WEST	43 36.44	53 44.92	22 18.64	118
TOTAL	111	209	98	418

(continued on next page)

(continued from previous page)

JANGRP	JULYGRP	ELECGRP	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
UNDER 32 DEGREES	UNDER 75 DEGREES	LOW	48	11.5	48	11.5
UNDER 32 DEGREES	UNDER 75 DEGREES	MEDIUM	70	16.7	118	28.2
UNDER 32 DEGREES	UNDER 75 DEGREES	HIGH	39	9.3	157	37.6
UNDER 32 DEGREES	75 DEGREES OR HI	LOW	15	3.6	172	41.1
UNDER 32 DEGREES	75 DEGREES OR HI	MEDIUM	22	5.3	194	46.4
UNDER 32 DEGREES	75 DEGREES OR HI	HIGH	14	3.3	208	49.8
32 DEGREES OR HI	UNDER 75 DEGREES	LOW	10	2.4	218	52.2
32 DEGREES OR HI	UNDER 75 DEGREES	MEDIUM	65	15.6	283	67.7
32 DEGREES OR HI	UNDER 75 DEGREES	HIGH	9	2.2	292	69.9
32 DEGREES OR HI	75 DEGREES OR HI	LOW	37	8.9	329	78.7
32 DEGREES OR HI	75 DEGREES OR HI	MEDIUM	84	20.1	413	98.8
32 DEGREES OR HI	75 DEGREES OR HI	HIGH	5	1.2	418	100.0

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 6

TABLE OF POPGRP BY CRIMEGRP

POPGRP(1980 POPULATION ESTIMATE) CRIMEGRP(CRIME RATE/100,000 POP (1981))

(10) FREQUENCY
(11) EXPECTED
(12) DEVIATION
CELL CHI2
ROW PCT

	6000 OR LESS	6001-750 0	7501-950 0	OVER 950 0	TOTAL
50,001-75,000	68	47	27	20	162
	(10) 42.2	39.5	41.5	38.8	
	(11) 25.8	7.5	-14.5	-18.8	
	(12) 15.7033	1.41115	5.04834	9.07697	
	41.98	29.01	16.67	12.35	
75,001-100,000	22	24	25	16	87
	22.7	21.2	22.3	20.8	
	-0.7	2.8	2.7	-4.8	
	0.02078	.361511	.334574	1.11317	
	25.29	27.59	28.74	18.39	
OVER 100,000	19	31	55	64	169
	44.1	41.2	43.3	40.4	
	-25.1	-10.2	11.7	23.6	
	14.261	2.54229	3.18556	13.74	
	11.24	18.34	32.54	37.87	
TOTAL	109	102	107	100	418

STATISTICS FOR TABLE OF POPGRP BY CRIMEGRP

STATISTIC	DF	VALUE	PROB
CHI-SQUARE	6	66.799	0.000
LIKELIHOOD RATIO CHI-SQUARE	6	68.810	0.000
MANTEL-HAENSZEL CHI-SQUARE	1	63.089	0.000
PHI		0.400	
CONTINGENCY COEFFICIENT		0.371	
CRAMER'S V		0.283	

SAMPLE SIZE = 418

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 7

TABLE OF INCOMGRP BY CRIMEGRP

INCOMGRP(PER CAPITA MONEY INCOME (1979)) CRIMEGRP(CRIME RATE/100,000 POP (1981))

FREQUENCY EXPECTED DEVIATION CELL CHI2 ROW PCT	6000 OR LESS	6001-750 0	7501-950 0	OVER 950 0	TOTAL
AVERAGE OR ABOVE	51 63.4 -12.4 2.41326 20.99	59 59.3 -0.3 .001484 24.28	62 62.2 -0.2 7E-04 25.51	71 58.1 12.9 2.84747 29.22	243
BELOW AVERAGE	58 45.6 12.4 3.35098 33.14	43 42.7 0.3 .002061 24.57	45 44.8 0.2 9E-04 25.71	29 41.9 -12.9 3.95391 16.57	175
TOTAL	109	102	107	100	418

STATISTICS FOR TABLE OF INCOMGRP BY CRIMEGRP

STATISTIC	DF	VALUE	PROB
CHI-SQUARE	3	12.571	0.006
LIKELIHOOD RATIO CHI-SQUARE	3	12.770	0.005
MANTEL-HAENSZEL CHI-SQUARE	1	11.271	0.001
PHI		0.173	
CONTINGENCY COEFFICIENT		0.171	
CRAMER'S V		0.173	

SAMPLE SIZE = 418

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 8

TABLE OF CITYGOVT BY CRIMEGRP

CITYGOVT(FORM OF CITY GOVERNMENT (1982)) CRIMEGRP(CRIME RATE/100,000 POP (1981))

FREQUENCY ROW PCT	6000 OR LESS	6001-750 0	7501-950 0	OVER 950 0	TOTAL
MAYOR COUNCIL	42 25.77	44 26.99	39 23.93	38 23.31	163
COUNCIL MANAGER	65 27.08	55 22.92	62 25.83	58 24.17	240
COMMISSION	2 13.33	3 20.00	6 40.00	4 26.67	15
TOTAL	109	102	107	100	418

CENSUS DATA: OUTPUT FROM FREQ PROCEDURE 9

STATISTICS FOR TABLE OF CITYGOVT BY CRIMEGRP

STATISTIC	DF	VALUE	PROB
CHI-SQUARE	6	3.295	0.771
LIKELIHOOD RATIO CHI-SQUARE	6	3.320	0.768
MANTEL-HAENSZEL CHI-SQUARE	1	0.527	0.468
PHI		0.089	
CONTINGENCY COEFFICIENT		0.088	
CRAMER'S V		0.063	

STATISTIC	VALUE	ASE
GAMMA	0.043	0.069

(continued on next page)

(continued from previous page)

```
KENDALL'S TAU-B                    0.027       0.043
STUART'S TAU-C                     0.025       0.040

SOMERS' D C|R                      0.032       0.052
SOMERS' D R|C                      0.022       0.036

PEARSON CORRELATION                0.036       0.048
SPEARMAN CORRELATION               0.030       0.048

LAMBDA ASYMMETRIC C|R              0.019       0.031
LAMBDA ASYMMETRIC R|C              0.000       0.000
LAMBDA SYMMETRIC                   0.012       0.020

UNCERTAINTY COEFFICIENT C|R        0.003       0.003
UNCERTAINTY COEFFICIENT R|C        0.005       0.005
UNCERTAINTY COEFFICIENT SYM        0.004       0.004

SAMPLE SIZE = 418
WARNING:  33% OF THE CELLS HAVE EXPECTED COUNTS LESS
          THAN 5. CHI-SQUARE MAY NOT BE A VALID TEST.

ASE IS THE ASYMPTOTIC STANDARD ERROR.
R|C MEANS ROW VARIABLE DEPENDENT ON COLUMN VARIABLE.
```

```
        CENSUS DATA: OUTPUT FROM FREQ PROCEDURE                          10

      SUMMARY STATISTICS FOR CITYGOVT BY CRIMEGRP


  COCHRAN-MANTEL-HAENSZEL STATISTICS (BASED ON TABLE SCORES)

STATISTIC   ALTERNATIVE HYPOTHESIS    DF       VALUE      PROB
---------------------------------------------------------------
    1       NONZERO CORRELATION       1       0.527     0.468
    2       ROW MEAN SCORES DIFFER    2       1.364     0.506
    3       GENERAL ASSOCIATION       6       3.287     0.772


TOTAL SAMPLE SIZE = 418
```

REFERENCES

County and City Data Book, 1983 Files on Tape Technical Documentation/ prepared by the Data User Services Division, Bureau of the Census. Washington: The Bureau, 1984.

County and City Data Book, 1983 Files on Tape (machine-readable data file)/ prepared by the Bureau of Census. Washington:The Bureau (producer and distributor), 1984.

Chapter 38

The MEANS Procedure

Operating systems: All

ABSTRACT

The MEANS procedure produces simple univariate descriptive statistics for numeric variables.

INTRODUCTION

PROC MEANS can compute statistics for an entire SAS data set or for groups of observations in the data set. If you use a BY statement, MEANS calculates descriptive statistics separately for groups of observations. Each group is composed of observations having the same values of the variables used in the BY statement. MEANS can also create one or more new SAS data sets containing the statistics calculated. If you want descriptive statistics in a data set and do not require printed output, use the SUMMARY procedure.

Other SAS procedures also compute univariate statistics. Although MEANS is the easiest and most direct descriptive procedure, other procedures provide more features. See UNIVARIATE, SUMMARY, CHART, and TABULATE.

Note: in this discussion, names of statistics refer to the sample estimates of the true parameters. Thus, the term *mean* refers to the sample mean, *variance* refers to the sample variance, and so forth.

SPECIFICATIONS

The procedure is controlled by the following statements:

PROC MEANS *options*;
VAR *variables*;
BY *variables*;
FREQ *variable*;
WEIGHT *variable*;
ID *variables*;
OUTPUT *options*;

There is no limit to the number of OUTPUT statements that can accompany a PROC MEANS statement.

PROC MEANS Statement

PROC MEANS *options*;

The options below can appear in the PROC MEANS statement:

DATA=*SASdataset*
: names the SAS data set to be analyzed by PROC MEANS. If DATA= is omitted, the most recently created SAS data set is used.

NOPRINT
: tells MEANS not to print any of the descriptive statistics. Use NOPRINT when the only purpose of using the procedure is to create a new SAS data set.

MAXDEC=*n*
: gives the maximum number of decimal places (0 to 8) for MEANS to use in printing results. Since the maximum field width for a result is 15, which includes two blank columns to separate values, MEANS may have to print fewer decimal places than the MAXDEC= value.

VARDEF=*divisor*
: specifies the divisor to be used in the calculation of the variance. VARDEF=DF requests that the degrees of freedom (N-1) be used as the divisor. VARDEF=WEIGHT or VARDEF=WGT requests that the sum of the weights be used. VARDEF=N requests that the number of observations (N) be used. VARDEF=WDF requests that the sum of the weights minus one be used. The default is VARDEF=DF.

The statistics below can be requested with the MEANS procedure by giving the keyword names of the statistics in the PROC MEANS statement. These keywords are also used in the OUTPUT statement described below. The statistics are defined in the chapter "SAS Univariate Procedures."

N	the number of observations on which calculations are based
NMISS	the number of missing values
MEAN	the mean
STD	the standard deviation
MIN	the smallest value
MAX	the largest value
RANGE	the range

SUM	the sum
VAR	the variance
USS	the uncorrected sum of squares
CSS	the corrected sum of squares
STDERR	the standard error of the mean
CV	the coefficient of variation (percent)
SKEWNESS	the measure of skewness
KURTOSIS	the measure of kurtosis
T	the Student's *t* value for testing the hypothesis that the population mean is zero
PRT	the probability of a greater absolute value of Student's *t*
SUMWGT	the sum of the WEIGHT variable values.

VAR Statement

VAR *variables;*

Statistics are calculated for each numeric variable listed in the VAR statement. If a VAR statement is not used, all numeric variables in the input data set, except for those listed in BY, ID, FREQ, or WEIGHT statements are analyzed. The results are printed in the order of the variables in the VAR statement.

BY Statement

BY *variables;*

A BY statement can be used with PROC MEANS to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in the chapter "Statements Used in the PROC Step."

FREQ Statement

FREQ *variable;*

When a FREQ statement appears with PROC MEANS, each observation in the input data set is assumed to represent *n* observations in the calculation of statistics, where *n* is the value of the FREQ variable. If the value of the FREQ variable is less than 1, the observation is not used in the calculations. If the value is not an integer, only the integer portion is used.

WEIGHT Statement

WEIGHT *variable;*

The WEIGHT statement specifies a numeric variable in the input SAS data set whose values are to be used to weight each observation. Only one variable can be specified. Both the FREQ and WEIGHT statements can be used. The WEIGHT variable values can be nonintegers and are used to calculate a weighted mean

and a weighted variance. If the value of the WEIGHT variable is less than zero or missing, a value of zero is assumed.

ID Statement

ID *variables;*

An ID statement can be used with PROC MEANS to include additional variables in the output data set. The value of any variable listed in an ID statement in the first observation of each BY group is included in the new data set.

OUTPUT Statement

OUTPUT *options;*

The OUTPUT statement requests that PROC MEANS output statistics to a new SAS data set. The options name the data set and the variables to be included.

OUT=*SASdataset*
: names the output SAS data set. If OUT= is not given, MEANS names the new data set according to the DATA*n* convention as if OUT=_DATA_ were specified. If you want to create a permanent SAS data set, you must specify a two-level name (see "SAS Files" for more information on permanent SAS data sets).

keyword=
names
: specifies the statistics you want in the new data set and names the variables containing these statistics. Any of the statistics available in the PROC MEANS statement can be output by using the keyword for that statistic. The keywords are listed below:

  N NMISS MEAN STD MIN MAX

  RANGE SUM VAR USS CSS STDERR

  CV SKEWNESS KURTOSIS T PRT SUMWGT

  The list of variable names after the equal sign names statistics for respective variables in the VAR statement. This list can be shorter than the VAR list.

For example, consider these statements:

```
PROC MEANS;
   VAR X1 X2;
   BY GROUP;
   OUTPUT OUT=STATS MEAN=MA MB STD=SA;
```

If the BY variable, GROUP, has two values A and B, the data set STATS contains two observations. Each of these observations contains the variables GROUP, MA, MB, and SA. MA and MB are the means of X1 and X2, respectively. SA is the standard deviation of X1. The standard deviation of X2 is not output. The statistics are printed since NOPRINT is omitted.

There is no limit to the number of OUTPUT statements that can accompany a PROC MEANS statement. Each OUTPUT statement creates one SAS data set.

DETAILS

Missing Values

PROC MEANS excludes missing values before calculating statistics. Each variable is treated individually: a missing value in one variable does not affect the calculations for other variables. Missing values for a BY variable form a separate BY group and are treated in the same way as BY groups with nonmissing values. If the FREQ variable value is missing, the observation is skipped. If the WEIGHT variable value is missing, the observation is treated as zero.

Output Data Set

The number of observations in the new data set corresponds to the number of BY groups for which statistics are calculated. If no BY statement is used, the output data set contains one observation. BY variables and ID variables as well as the computed statistics are included in the new data set.

Printed Output

When no statistics are specifically requested in the PROC MEANS statement, MEANS prints these statistics for each variable in the VAR statement:

1. the name of the variable
2. N, the number of nonmissing values for the variable
3. the MEAN or average of the variable
4. the STANDARD DEVIATION of the variable
5. the MINIMUM VALUE of the variable
6. the MAXIMUM VALUE of the variable.

If there is space available on the output (depending on the computer terminal used and LINESIZE= specified), MEANS also prints these statistics:

7. STD ERROR OF MEAN, the standard error of the mean
8. the SUM of all the values for a variable
9. the VARIANCE of each variable
10. C.V., or the coefficient of variation expressed as a percentage.

If statistics are specifically requested on the PROC statement, only those statistics are printed. In addition to the statistics above, upon request MEANS can print these statistics:

11. N MISSING, the number of missing values for each variable
12. the RANGE of each variable
13. the UNCORRECTED SS, the raw sum of squares (not adjusted for the mean)
14. the CORRECTED SS, the sum of squares adjusted for the mean
15. SKEWNESS
16. KURTOSIS
17. T, the Student's *t* value for testing the hypothesis that the population mean is zero
18. PR > |T|, the probability of a greater absolute value for Student's *t* under the hypothesis that the mean is zero
19. SUMWGT, the sum of the WEIGHT variable values.

If a statistic cannot be computed because of insufficient or inappropriate data, the statistic is reported as missing. See the chapter "SAS Univariate Procedures."

If a BY statement is included, a line is drawn to separate the statistics for each BY group.

EXAMPLES

Univariate Statistics: Example 1

The following example requests univariate statistics for two variables, RATING and EXCESS. The first PROC MEANS statement asks for statistics on all numeric variables (including DAY) for all observations.

The second PROC MEANS statement asks for the statistics that must be specifically requested. In this small sample, statistics like skewness and kurtosis are not reliable.

The third PROC MEANS statement requests statistics for each combination of values of the variables PLACE and DAY. MAXDEC=3 requests that only three decimal places be used to print the results. MEANS creates an output data set that includes the means and standard errors of the means for the two variables. The PRINT procedure is used to print the new data set.

```
DATA A;
   INPUT RATING EXCESS PLACE $ DAY @@;
   CARDS;
04 54 S 1   07 70 N 1   10 69 N 2   04 52 S 1
07 70 S 2   08 74 N 1   04 60 S 1   07 62 S 2
07 80 N 1   06 61 S 2   06 77 N 2   08 75 N 2
;
PROC MEANS;
  TITLE 'OUTPUT FROM MEANS PROCEDURE';

PROC MEANS DATA=A MAXDEC=3 NMISS RANGE USS CSS
      SKEWNESS KURTOSIS T PRT SUMWGT;
   VAR RATING EXCESS;
   TITLE 'REQUESTED STATISTICS';

PROC SORT;
   BY PLACE DAY;
PROC MEANS MAXDEC=3;
   BY PLACE DAY;
   VAR RATING EXCESS;
   OUTPUT OUT=NEW MEAN=RMEAN EMEAN STDERR=RSE ESE;
   TITLE 'STATISTICS BY PLACE AND DAY';
PROC PRINT;
   TITLE 'NEW DATA SET';
```

Output 38.1 Requesting All Numeric Variables

OUTPUT FROM MEANS PROCEDURE 1

❶ VARIABLE	❷ N	❸ MEAN	❹ STANDARD DEVIATION	❺ MINIMUM VALUE	❻ MAXIMUM VALUE	❼ STD ERROR OF MEAN	❽ SUM	❾ VARIANCE	❿ C.V.
RATING	12	6.50000000	1.83402191	4.00000000	10.00000000	0.52943652	78.0000000	3.36363636	28.216
EXCESS	12	67.00000000	9.08545291	52.00000000	80.00000000	2.62274434	804.0000000	82.54545455	13.560
DAY	12	1.50000000	0.52223297	1.00000000	2.00000000	0.15075567	18.0000000	0.27272727	34.816

Output 38.2 Requesting Assorted Statistics

REQUESTED STATISTICS 1

VARIABLE	⓫ N MISSING	⓬ RANGE	⓭ UNCORRECTED SS	⓮ CORRECTED SS	⓯ SKEWNESS	⓰ KURTOSIS	⓱ T	⓲ PR>|T|	⓳ SUM OF WEIGHTS
RATING	0	6.000	544.000	37.000	0.053	-0.165	12.28	0.0001	12.000
EXCESS	0	28.000	54776.000	908.000	-0.312	-1.056	25.55	0.0001	12.000

Output 38.3 Requesting Statistic by Two Variables

STATISTICS BY PLACE AND DAY 1

VARIABLE	N	MEAN	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	STD ERROR OF MEAN	SUM	VARIANCE	C.V.
PLACE=N DAY=1									
RATING	3	7.333	0.577	7.000	8.000	0.333	22.000	0.333	7.873
EXCESS	3	74.667	5.033	70.000	80.000	2.906	224.000	25.333	6.741
PLACE=N DAY=2									
RATING	3	8.000	2.000	6.000	10.000	1.155	24.000	4.000	25.000
EXCESS	3	73.667	4.163	69.000	77.000	2.404	221.000	17.333	5.652
PLACE=S DAY=1									
RATING	3	4.000	0.000	4.000	4.000	0.000	12.000	0.000	0.000
EXCESS	3	55.333	4.163	52.000	60.000	2.404	166.000	17.333	7.524
PLACE=S DAY=2									
RATING	3	6.667	0.577	6.000	7.000	0.333	20.000	0.333	8.660
EXCESS	3	64.333	4.933	61.000	70.000	2.848	193.000	24.333	7.668

NEW DATA SET 2

OBS	PLACE	DAY	RMEAN	EMEAN	RSE	ESE
1	N	1	7.33333	74.6667	0.33333	2.90593
2	N	2	8.00000	73.6667	1.15470	2.40370
3	S	1	4.00000	55.3333	0.00000	2.40370
4	S	2	6.66667	64.3333	0.33333	2.84800

T Test for Paired Comparisons: Example 2

See the description of the TTEST procedure in the *SAS User's Guide: Statistics* for an example of using PROC MEANS to compute a *t* for paired comparisons.

Chapter 39

The OPTIONS Procedure

Operating systems: All

ABSTRACT

The OPTIONS procedure lists the current values of all global SAS system options. While the OPTIONS statement (described in the "SAS Statements Used Anywhere" chapter) sets SAS system options specifications during a SAS job, the OPTIONS procedure simply reports the setting of the SAS system options currently in effect.

INTRODUCTION

The global SAS system options are used to control certain SAS data library and SAS data set attributes, SAS output features, the efficiency of program execution, and so on. You can usually rely on the default values for the global SAS system options instead of specifying them explicitly. (The default values of SAS system options are set by your installation rather than by the SAS System.) To learn more about the global SAS system options, consult the description of the OPTIONS statement in the "SAS Statements Used Anywhere" chapter and the "SAS System Options" chapter.

PROC OPTIONS prints a list of the global SAS system options, giving default settings or settings you have specified for the options in an OPTIONS statement or in the control language invoking SAS.

SPECIFICATIONS

The only statement used with PROC OPTIONS is

PROC OPTIONS *options*;

PROC OPTIONS Statement

PROC OPTIONS *options*;

The following options can be used in the PROC OPTIONS statement:

SHORT requests an abbreviated listing of the system options. If SHORT is not specified, each option is listed on a separate line and includes an explanation.

CMS lists the SAS system options specific to the SAS System under CMS by using the CMS option.

DLI IMS lists the SAS system options specific to SAS/IMS-DL/I.

Additional options are defined for use during SAS installation and for use when other SAS software products are installed. They are documented in their respective installation instructions and product user's guides.

DETAILS

Printed Output

PROC OPTIONS prints a list titled SYSTEM PARAMETERS AND OPTIONS on the SAS log. The output is in two columns if SHORT is not specified, with option names and values in the left column and explanations in the right column. If the SHORT option is specified, the option names and their values are listed in paragraph format.

EXAMPLE

PROC OPTIONS Default Output

This example shows the default output from PROC OPTIONS:

```
PROC OPTIONS;
```

Output 39.1 Output Produced Using PROC OPTIONS

```
1      S A S   L O G   OS SAS 5.XX          VS2/MVS JOB OPTIONX  STEP SASBASE

NOTE: COPYRIGHT (C) 1984 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB OPTIONX HAS BEEN RUN UNDER RELEASE 5.05 OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
      SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.37 SECONDS.

1          PROC OPTIONS;

SYSTEM PARAMETERS AND OPTIONS

BAUD=1200           COMMUNICATIONS LINE BIT RATE
BLKSIZE=0           DEFAULT BLKSIZE
BUFNO=2             DEFAULT NUMBER OF BUFFERS
BYERR               NULL BY LIST/STMT ON _NULL_ INPUT DATA SETS AN ERROR?
NOCAPS              TRANSLATE QUOTED STRINGS AND TITLES TO UPPER CASE?
CARDS=MAX           CARDS PUNCHED LIMIT
C96                 EITHER C48, C60 OR C96 FOR 48, 60 OR 96 CHARACTER SET
CENTER              CENTERING?
NOCHARCODE          CHARACTER CODES ?-,?=,?/ FOR _,~,|
```

(continued on next page)

(continued from previous page)

```
NOCHKPT             CHECKPOINT AT END OF EVERY STEP?
CMDMAC              SAS STATEMENTS SCANNED FOR IMPLICIT INVOCATION OF A
                    "CMND" MACRO?
DATE                DATE PRINTED IN TITLE?
DEFAULT=LOCAL       DEFAULT SET OF OPTIONS USED
DEVADDR=(SAS001 . PRINTER1)
                    DEVICE ADDRESS, QNAME OR NODE NAME
DEVICE=             TERMINAL DEVICE NAME
NODMS               DISPLAY MANAGEMENT SYSTEM REQUESTED?
DQUOTE              ALLOW DOUBLE-QUOTED (QUOTATION MARKS:"") LITERALS?
DSNFERR             TREAT DATA SET NOT FOUND AS AN ERROR? NO = SET TO _NULL_
DSRESV              OS DATA SETS RESERVED OR ENQUEUED?
NODUMP              SAS ERROR HANDLER INVOKED?
NODUMPSYS           ALLOW INTERCEPTION OF PROGRAM CHECKS AND ABENDS?
NODYNALLOC          SORT UTILITY TO DYNAMICALLY ALLOCATE SORT WORK AREAS?
NOERRORABEND        ABEND ON ERROR CONDITIONS?
ERRORS=20           MAXIMUM NUMBER OF OBSERVATIONS WITH ERROR MESSAGES
FILSZ               SORT UTILITY SUPPORT FILSZ PARAMETER?
FIRSTOBS=1          DEFAULT FIRST OBSERVATION TO BE PROCESSED
FMTERR              TREAT MISSING FORMAT OR INFORMAT AS AN ERROR?
NOFS                PROCEDURES TO OPERATE IN FULL SCREEN MODE?
FSP                 SAS/FSP ENABLED?
GRAPHICS            SAS/GRAPH SUPERVISOR REQUIRED?
IMPLMAC             SAS STATEMENTS SCANNED FOR IMPLICIT INVOCATION OF A
                    'STMT' MACRO?
IMS                 SAS/IMS-DL/I ENABLED?
INCLUDE             PROCESS %INCLUDE STATEMENT(S)?
INITSTMT=''         INITIAL SAS STATEMENT(S) EXECUTED BEFORE SYSIN OR SYSIPT
INTERACTIVE         DO NOT TAKE BATCH DEFAULTS WHEN EXECUTING INTERACTIVELY
INVALIDDATA=.       MISSING VALUE FOR INVALID DATA
LABEL               PRINT VARIABLE LABELS?
_LAST_=_NULL_       LAST DATA SET CREATED
LEAVE=0             UNALLOCATED MEMORY PARAMETER
LINESIZE=132        LINE SIZE FOR PRINT
LISTPF=(LEFT RIGHT CURSOR BACK FORWARD END HELP TOP BOTTOM RETURN FIND CMD HELP RETURN END BOTTOM FIND CMD BACK FORWARD TOP LEFT
```

```
2      S A S  L O G   OS SAS 5.XX          VS2/MVS JOB OPTIONX  STEP SASBASE

RIGHT CURSOR)       DEFAULT PF KEY DEFINITIONS FOR HELP FACILITY (AND FOR
                    PROC FSLIST)
LOG=FT11F001        DDNAME FOR LOG
MACRO               PERFORM MACRO PROCESSING?
NOMACROGEN          LIST MACRO TEXT GENERATED?
MAUTOSOURCE         SAS MACRO AUTOMATIC CALL ALLOWED?
NOMCOMPILE          MACRO COMPILER LOADED BY DEFAULT?
NOMEMERR            ABEND IF MEMORY-EXCEEDED ERROR OCCURS?
MEMRPT              MEMORY USAGE REPORTS DISPLAYED?
MERROR              APPARENT UNDEFINED MACRO REFERENCES CONSIDERED AN ERROR?
NOMEXTMOD           EXECUTION OF EXTERNAL MODULES ALLOWED FROM MACRO
                    FACILITY?
MISSING='.'         PRINTED SYSTEM MISSING VALUE
MLEAVE=6144         UNALLOCATED MACRO MEMORY PARAMETER?
NOMLOGIC            MACRO LOGIC TRACE OPTION?
MODECHARS='?>*'     PROMPT MODE CHARACTERS
NOMPRINT            MACRO FACILITY RESULTS PRINTED IN A SYNTHETIC COMPRESSED
                    FORM?
NOMRECALL           LOOK UP AUTOCALL MACROS AND EXTERNAL MODULES EVERYTIME?
MSIZE=12288         AMOUNT OF MEMORY RESERVED FOR MACRO FACILITY
MSYMSIZE=1024       INITIAL MACRO SYMBOL TABLE SIZE
MWORK=2048          MACRO WORK AREA SIZE
NONEWS              CURRENT NEWS PRINTED ON LOG?
NOTES               SAS NOTES ARE TO BE PRINTED.
NUMBER              PAGES NUMBERED?
OBS=MAX             DEFAULT LAST OBSERVATION TO BE PROCESSED
OFFLINE=0.0040      COST OF OFFLINE DISK STORAGE PER TRACK PER DAY
ONLINE=0.0120       COST OF ONLINE DISK STORAGE PER TRACK PER DAY
OPLIST              LIST EXEC OPTIONS ON LOG?
OVP                 NOOVP IF NO OVERPRINTING ALLOWED, OVP OTHERWISE
PAGES=MAX           PAGES PRINTED LIMIT
PAGESIZE=60         PAGE SIZE FOR PRINT
PARM=''             PARM FOR EXTERNAL PROGRAMS
PARMCARDS=FT15F001  DDNAME FOR PARMCARDS
PRINTDEVICE=SYSOUT  SAS PRINT FILE OUTPUT ULTIMATE DESTINATION DEVICE NAME
PRINTINIT           SAS PRINT FILE INITIALIZED?
PROBSIG=0           NUMBER OF SIGNIFICANT FIGURES GUARANTEED WHEN
                    PRINTING P-VALUES
PROCSIZE=16776192   MAXIMUM PROCEDURE MEMORY ALLOCATION
PROMPTCHARS='000A010D05000000'X
                    TERMINAL PROMPT CHARACTERS FOR SAS/GRAPH
REPLACE             ALLOW REPLACE OF PERMANENTLY ALLOCATED SAS DATA SETS?
S=0                 SOURCE STATEMENT LENGTH
```

(continued on next page)

(continued from previous page)

```
S2=S                %INCLUDE STATEMENT LENGTH
S370                IS MACHINE A SYSTEM 370, 303X, 43XX, OR 308X?
SASAUTO=SASAUTO     DDNAMES FOR AUTOMATIC CALL OF PRECOMPILED MACROS
SASAUTOS=SASAUTOS   AUTOMATIC CALL LIBRARY FOR SOURCE MACROS
SASHELP=SASHELP     DDNAME OF NEWS/HELP STATEMENT LIBRARY
SASNEWS=SASNEWSB    MEMBER OF HELP LIBRARY TO PRINT WHEN NEWS OPTION
                    SPECIFIED
SEQ=8               NUMBER OF NUMERIC DIGITS IN SEQUENCE NUMBERS
SERROR              APPARENT UNDEFINED SYMBOLIC REFERENCES CONSIDERED
                    AN ERROR?
SKIP=0              NUMBER OF LINES TO SKIP BEFORE TITLE
NOSNP               SNAP MACROS ENABLED?
NOSNPPROG           SNAP PROGRAM DATA SET?
SORT=4              MINIMUM SIZE OF SORT WORK AREAS IN CYLINDERS
SORTDEV=RIO         DEVICE NAME USED TO DYNAMICALLY ALLOCATE SORT WORK AREAS
```

```
3      S A S   L O G   OS SAS 5.XX          VS2/MVS JOB OPTIONX  STEP SASBASE

SORTLIB=''          SORT LIBRARY DSNAME
NOSORTLIST          PASS LIST OPTION TO SORT?
NOSORTMSG           PASS MSG OPTION TO SYSTEM SORT UTILITY?
SORTPGM='SORT'      ENTRY POINT NAME OF SYSTEM SORT
SORTSIZE=MAX        SIZE PARAMETER FOR THE SYSTEM SORT
SORTWKDD='SASS'     PREFIX OF DYNAMICALLY ALLOCATED SORT WORK AREA DDNAMES
SORTWKNO=3          DEFAULT NUMBER OF SORT WORK AREAS TO ALLOCATE
SOURCE              LIST SAS SOURCE STATEMENTS?
SOURCE2             LIST INCLUDED SAS SOURCE STATEMENTS?
NOSPOOL             SPOOL TERMINAL INPUT?
STIMER              SAS ALLOWED TO USE SCP TASK TIMING FACILITIES?
NOSYMBOLGEN         PRINT SYMBOLIC REPLACEMENT TEXT?
NOSYNCSORT          SYSTEM SORT IS SYNCSORT?
SYSIN=SYSIN         DDNAME FOR PRIMARY INPUT
SYSIN               READ STATEMENTS FROM PRIMARY INPUT FILE?
SYSPARM=''          VALUE TO BE RETURNED BY SYSPARM() FUNCTION
TAPE=TAPE           TAPE UNIT NAME
TAPECLOSE=REREAD    CLOSE DISPOSITION (VOLUME POSITIONING) FOR TAPE
                    DATA LIBRARIES
TEXT82              TITLE/FOOTNOTE COMPATIBLE WITH VERSION 82
TIME=MAX            TIME LIMIT IN SECONDS (ELAPSED TIME UNDER DOS/VSE)
TLS=0               LINE SIZE FOR TERMINAL
TPS=0               PAGE SIZE FOR TERMINAL
TRANTAB=GTABVTAM    TERMINAL TRANSLATE TABLE
NOTSO               IS SAS TO PRODUCE RESOURCE UTILIZATION NOTES?
UNITS=11 12 13 14 15 16 17 18 19 20
                    SAS DD ASSIGNMENTS/LOGICAL UNIT NUMBERS
USER=WORK           DDNAME FOR USER
USERPARM=''         PARM TEXT AVAILABLE TO USER EXITS
VNFERR              TREAT VARIABLE NOT FOUND ON _NULL_ DATA SET AS ERROR?
VSAMLOAD            LOADING OF VSAM FILES ALLOWED?
VSAMREAD            READING OF VSAM FILES ALLOWED?
VSAMUPDATE          UPDATE OF VSAM FILES ALLOWED?
WORK=WORK           DDNAME FOR WORK
WORKINIT            INITIALIZE WORK FILE?


DEVICE TYPE  2301   2302   2303  2305-1 2305-2  2311   2314
-----------  -----  -----  -----  -----  -----  -----  -----
    BLKSIZE  20483   4984   4892  14136  14660   3625   7294
FILEBLKSIZE   6720   4960   4880   6800   7200   3600   3520

DEVICE TYPE  2321   3330  3330-1  3340   3350   3375   3380
-----------  -----  -----  -----  -----  -----  -----  -----
    BLKSIZE   2000  13030  13030   8368  19069  17600  23476
FILEBLKSIZE   2000   6400   6400   4080   6160   6800   9040

DEVICE TYPE  2400   3400  SYSOUT  CMS    OTHER  FBA
-----------  -----  -----  -----  -----  -----  -----
    BLKSIZE  20480  32760       .  32760  10240  20480
FILEBLKSIZE   6400   8000     141  32760   6400      .


PRINTDEV  FORMCHAR(PRINTDEVICE)
--------  ----------------------------------------
STANDARD  ('4F606060604F4E4F6060604E7E4F6061E04C6E5C'X)
TEXT      ('4FBFACBFBC4F8F4FABBFBB4E7E456061E04C6EAF'X)
IBM6670   ('FABFACCCBCEB8FECABCBBB4E7E4F6061E04C6EAF'X)
```

```
4      S A S  L O G   OS SAS 5.XX          VS2/MVS JOB OPTIONX  STEP SASBASE
0IBM3800   ('4FBFACBFBC4F8F4FABBFBB'X)


NOTE: THE PROCEDURE OPTIONS USED 0.41 SECONDS AND 676K.
      THE COMPILE   PHASE USED 0.37 SECONDS.
      THE EXECUTION PHASE USED 0.05 SECONDS.

NOTE: SAS USED 676K MEMORY.
NOTE: THE COMPILE   PHASE USED 0.37 SECONDS.
      THE EXECUTION PHASE USED 0.05 SECONDS.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Chapter 40

The PDS Procedure

Operating system: OS

ABSTRACT

The PDS procedure can list, delete, and rename the members of an OS partitioned data set.

INTRODUCTION

Partitioned data sets are libraries containing files (called members) on OS operating systems. These libraries are used to store program source code, macros, cataloged procedures, load modules, and other data. PROC PDS operates on the directory of a partitioned data set to list, delete, and rename members and aliases. (Partitioned data sets are not the same as SAS data libraries.)

SPECIFICATIONS

PROC PDS DDNAME=*fileref options*;
  DELETE *member ...*;
  CHANGE *oldname=newname oldname=newname ...*;
  EXCHANGE *name=anothername ...*;

PROC PDS Statement

PROC PDS DDNAME=*fileref options;*

DDNAME=*fileref* specifies the *fileref* (DDname) of the JCL DD statement or the filename assigned in the TSO ALLOC command defining the partitioned data set to be processed. The DDNAME= specification must appear.

The options below can appear in the PROC PDS statement:

NOLIST suppresses the listing of the member names and aliases in the directory of the partitioned data set.

KILL deletes all the members of the partitioned data set specified by DDNAME=.

DELETE Statement

DELETE *member ...;*

If you want to delete a member or members from the PDS, specify their names in a DELETE statement.

When a specification in the DELETE statement is followed by a colon (:), all members whose names begin with the characters preceding the colon are deleted. For example, when this statement is executed:

```
DELETE PRGM:;
```

PROC PDS deletes all members whose names begin with the characters PRGM.

CHANGE Statement

CHANGE *oldname=newname ...;*

If you want to rename a member or members of the partitioned data set, use the CHANGE statement. Specify the old name on the left side of the equal sign, the new name on the right. For example, the statements:

```
PROC PDS DDNAME=LIB;
   CHANGE TESTPGM=PRODPGM;
```

change the name of member TESTPGM to PRODPGM.

If there are multiple members whose names begin with the same sequence of characters and you want to change all of the names so they begin with a different sequence, use the colon (:) notation after the *oldname* and *newname*. Here is an example:

```
CHANGE EXAM:=TEST:;
```

All of the members that had names beginning with the characters EXAM now have names beginning with the characters TEST.

It is not necessary for the lengths of the sequence of characters specified with colon notation to match. For example:

```
CHANGE AM:=MORN:;
```

EXCHANGE Statement

EXCHANGE *name=anothername...;*

Use the EXCHANGE statement to interchange the names of members of the partitioned data set. For example, after these statements are executed:

```
PROC PDS DDNAME=MYLIB;
   EXCHANGE A=Z;
```

the member of MYLIB originally called A is now Z, and the member originally called Z is now A.

If there are multiple members whose names begin with the same sequence of characters and you want to exchange that sequence with the sequence from another group of data sets, use the colon (:) notation after *name* and *anothername*. For example, when this statement is executed:

```
EXCHANGE ABC:=DEFG:;
```

all data sets that originally began with ABC now begin with DEFG. All of the data sets that began with DEFG now begin with ABC.

It is not necessary for the lengths of the sequence of characters specified with colon notation to match.

DETAILS

Output Data Set

The PDS procedure updates the directory of the partitioned data set when members are deleted or renamed.

Usage Note

Unlike other SAS procedures that deal with partitioned data sets (for example, PDSCOPY and SOURCE), PROC PDS does not make any distinction between a member name and an alias, other than to report which names in the PDS directory are aliases for which members. If an alias is renamed, it is still an alias. PROC PDS allows you to delete a member that has aliases in the PDS directory, but then other procedures (PROC PDSCOPY, for example) cannot process the aliases.

Printed Output

Unless NOLIST is specified, PROC PDS prints an updated listing of the members in the partitioned data set. The list on the SAS log reflects member name changes and deletions.

EXAMPLE

Deleting and Renaming Members with PROC PDS

The example below lists the contents of the partitioned data set XXXXXX.YYYYYY.ZZZZ and then prints a second listing showing member changes and deletions specified by the second PROC step.

```
PROC PDS DDNAME=LIB;
PROC PDS DDNAME=LIB;
   DELETE IMS: ERA MARQ;
   CHANGE TEXT=CONJOB;
```

Output 40.1 Deleting and Renaming Members with PROC PDS

```
1         SAS(R) LOG   OS SAS 5.XX           MVS/XA JOB SASTEST  STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.        (XXXXXXXX).

NOTE: THE INITIALIZATION PHASE USED 0.74 SECONDS.
1
2              PROC PDS DDNAME=LIB;
3

           DSNAME=XXXXXX.YYYYYYY.ZZZZ,VOL=SER=QQQ333

           MEMBERS

           A107 CAA COST CREDITS ERA FSP7 IMS IMSAN IMSI IMSX MARQ NEWAN RUN SASJOB TEXT UP

             45 TRACKS ALLOCATED
              2 TRACKS USED
             43 TRACKS UNUSED
              1 EXTENTS
             30 DIRECTORY BLOCKS ALLOCATED
              3 DIRECTORY BLOCKS USED

NOTE: THE PROCEDURE PDS USED 0.72 SECONDS AND 336K.

3              PROC PDS DDNAME=LIB;
4                 DELETE IMS: ERA MARQ;
5                 CHANGE TEXT=CONJOB;

           DSNAME=XXXXXX.YYYYYYY.ZZZZ,VOL=SER=QQQ333

           MEMBERS

           A107 CAA CONJOB COST CREDITS FSP7 NEWAN RUN SASJOB UP

             45 TRACKS ALLOCATED
              2 TRACKS USED
             43 TRACKS UNUSED
              1 EXTENTS
             30 DIRECTORY BLOCKS ALLOCATED
              2 DIRECTORY BLOCKS USED

NOTE: THE PROCEDURE PDS USED 0.70 SECONDS AND 336K.
NOTE: SAS USED 336K MEMORY.
```

```
2         SAS(R) LOG   OS SAS 5.XX           MVS/XA JOB SASTEST  STEP SASTEST

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Chapter 41

The PDSCOPY Procedure

Operating system: OS

ABSTRACT

The PDSCOPY procedure copies OS partitioned data sets containing load modules from disk to disk, disk to tape, tape to tape, or tape to disk.

INTRODUCTION

The PDSCOPY procedure can be used to copy an entire library, or you can specify that only certain modules be copied. PROC PDSCOPY is useful for backing up load module libraries to tape. If you use PROC PDSCOPY to copy a library to tape, you must also use it if you want to copy that library back to disk. You can use either PROC PDSCOPY or a utility program to copy a PDSCOPY tape to another tape. When libraries are moved between disks with different optimal block sizes, PROC PDSCOPY can be used to reblock the libraries. PROC PDSCOPY handles overlay programs and alias names. It also sets up the RLD count fields used by program FETCH on MVS/XA and MVS systems.

Modules that are copied with PROC PDSCOPY take 13% to 18% less disk space after copying because PROC PDSCOPY uses space left on a partially filled track to store records. The linkage editor constructs records that do not fit on a partially used track.

The PDSCOPY procedure does not copy scatter-loaded modules.

If errors are encountered during PDSCOPY processing, the return code for the job step is set to 8.

SPECIFICATIONS

PROC PDSCOPY INDD=*fileref* **OUTDD=***fileref options*;

SELECT *modulename ...;*
EXCLUDE *modulename ...;*

PROC PDSCOPY Statement

PROC PDSCOPY INDD=*fileref* OUTDD=*fileref options;*

INDD=*fileref*
specifies the *fileref* of the load module library to be copied. INDD= must be specified.

OUTDD=*fileref*
specifies the *fileref* of the output partitioned data set. OUTDD= must be specified.

The options that can appear in the PROC PDSCOPY statement are listed below:

ALIASMATCH=TTR | NAME | BOTH | EITHER
specifies how aliases are to be matched with main modules to determine if they represent the same module.

- TTR, the default, specifies that TTR (relative track and record) values are compared. An alias is assumed to represent the main module with the same TTR value. If the TTR values match, the directory entry for the main module and the alias currently point to the same place in the data set. TTR corresponds to the method used by previous releases of PROC PDSCOPY.

  The most common situation in which TTR values might match, but names may not, occurs when the main module has been renamed (for example, by using SPF option 3.1) since the aliases were created. The alias directory entries may still contain the old main module name.
- NAME specifies that the "main member name" in the directory entry for the alias (at offset 36) is compared with main module names to find a match. The alias is assumed to represent the same module as the main module with the matching name. When you specify NAME, the TTR values do not need to match.

  A situation in which names match while TTR values do not occurs when the main module is originally link edited specifying the alias names, and later link edited again without specifying them. In this case, the directory entries for the aliases still point to the older version of the module (that is, to a back-level version). Because of this, you should consider carefully whether to use NAME, which updates the aliases to point to the current version of the main module rather than the back-level version, or use the NEWMOD option, which causes the older version of the module to be copied, or simply use TTR matching and not copy the aliases when they are, in fact, obsolete names.
- BOTH specifies that both the TTR values and the names must match as described above before a main module is considered a match.
- EITHER specifies that a match for either the TTR value **or** the name, as described above, is sufficient to identify the main module corresponding to an alias. If more than one main module directory entry matches, it is impossible to predict which one will be used.

Whichever method is being used, unmatched aliases are not copied to the output file unless you specify the NEWMOD option

(described below). Matched aliases in the output file always point to the main module to which they were matched (that is, they have the same TTR values), even if the TTR values were different in the input file (which might occur if NAME or EITHER was used). However, when modules are matched using the TTR values (that is, TTR or EITHER was specified), the "main member name" in alias directory entries is not changed in the output file; it remains the same as it was in the input file, even if the name did not match the name of the main module with the same TTR.

BLKSIZE=*b*

specifies the maximum block size to be used for reblocking the copied load modules on the output device. If BLKSIZE= is omitted, the default depends on the device type of the output device. The default block size varies depending on the device and data set type:

- If output is to tape, the default is 32760.
- If output is in sequential format on disk (that is, the OUTTAPE option was used), the default is the device track size or 32760, whichever is less.
- If output is to disk, the default depends on the device type, but it is never larger than 18K (18432) or the device track size.
- Unless the NODCBS option is specified (see below) and the output data set is a partitioned data set on disk, the default value is reduced to the data set control block (DSCB) block size of the partitioned data set, if that is smaller.

For tape (sequential) format output, the specified block size cannot be less than 1.125 times the maximum input device block size, nor greater than 32760. For disk output, the specified block size cannot be less than 1024. Also for disk, the block size cannot be greater than 18K unless you use the MAXBLOCK parameter (see below) and, in any case, cannot exceed the device track size or 32760, whichever is less.

DC

specifies that load modules marked "downward compatible" (that is, modules that can be processed by pre-OS/VS linkage editors) are eligible for processing. After they are copied by PROC PDSCOPY, the load modules are not marked "DC" in their directory entry because PROC PDSCOPY does not produce downward compatible load modules or preserve their attributes. If you do not specify the DC option and you attempt to copy load modules marked DC, PROC PDSCOPY issues an error message.

DCBS

specifies that the data control block (DCB) characteristics of the output partitioned data set on disk are to be preserved. This option is now the default, but it is preserved for compatibilty with earlier releases of PROC PDSCOPY. See the description of the NODCBS option, below.

INTAPE

specifies that the INDD= library is in tape (sequential) format. INTAPE is assumed if the device allocated to the input data set is a tape drive.

MAXBLOCK=*b*
MAXBLK=*b*

allows you to override the limitation of 18K on the block size of text records in the output library. (BLKSIZE must be GE MAXBLK to get

text records at MAXBLK size.) If MAXBLOCK is not specified, the maximum block size for text records is 18K; this is the largest size of text block that can be handled by FETCH on most OS systems. You can specify a block size for text records greater than 18K, but doing so may cause copied modules to abend with an abend code of 0C4 or 106-E when they are executed. We recommend that you use this parameter only if you are sure that your operating system (or TP monitor) FETCH routines support text blocks larger than 18K. CICS program FETCH, for example, will support text blocks larger than 18K.

NE

specifies that the output library should not contain records used in the link editing process. Although programs in the output library are executable, they cannot be reprocessed by the linkage editor, nor can they be modified by the IMASPZAP program. Using the NE option can reduce disk space required for the output library by an additional 10% to 20%.

NEWMOD

specifies that aliases not matching a main module are to be copied as main modules in their own right. These names are not marked as aliases in the output file. The directory entry on the output file is reformatted to main module format. See the ALIASMATCH option for a description of how alias names are matched with main module names. If you do not specify NEWMOD, unmatched aliases are not copied to the output file.

NOALIAS
NOA

prevents automatic copying of all aliases of each module selected for copying. Any aliases that you want to be copied must be specifically selected. Note that if you select only an alias of a module, the module (that is, the main module name) is still automatically copied, along with the selected alias.

NODCBS

specifies that the data control block (DCB) characteristics of the output partitioned data set on disk can be overridden. (This option is the converse of the DCBS option, described above.) If the NODCBS option is specified, PROC PDSCOPY changes the DSCB block size of the output partitioned data set to the maximum permissible block size for the device. Otherwise, the maximum permissible value of the BLKSIZE= option is the current block size value from the DSCB, and the DSCB blocksize is not changed.

Using the NODCBS option may enable PROC PDSCOPY to block output load modules more efficiently. However, changing the DSCB blocksize could cause problems when the data set is moved, copied, or backed up by a program other than PROC PDSCOPY, particularly if your installation has more than one type of disk drive. Consult your systems staff before specifying NODCBS.

NOREPLACE
NOR

copies only members in the INDD= library that are not found in the OUTDD= library; that is, members or aliases with the same name are not replaced.

OUTTAPE
: specifies that the OUTDD= library is to be in tape (sequential) format. OUTTAPE is assumed if the device allocated to the output data set is a tape drive.

SHAREINPUT
SHAREIN
: specifies that the INDD= library is to be shared with other jobs and TSO users. By default, both the INDD= library and the OUTDD= library are enqueued for exclusive control against SPF and the linkage editor so that they cannot be changed while PROC PDSCOPY is processing them. If you specify the SHAREINPUT option, the INDD= library is shared with SPF and the linkage editor rather than enqueued exclusively. (The OUTDD= library is always enqueued exclusively). Use this option if you want to continue to use the INDD= library elsewhere while you are processing it with PROC PDSCOPY. Take care that the modules you are copying are not changed or replaced.

SELECT Statement

SELECT *modulename ...;*

Use the SELECT statement to specify the names of modules to be copied if you do not want to copy the entire library. You can specify as many SELECT statements as necessary.

If you follow a specification in the SELECT statement with a colon (:), all modules beginning with the characters preceding the colon are included in the copy operation. In the following example:

```
SELECT FCS:;
```

all data sets with names that begin with the characters FCS are copied.

EXCLUDE Statement

EXCLUDE *modulename ...;*

Use the EXCLUDE statement if you want to exclude certain modules from the copying operation. The EXCLUDE statement is useful if you want to copy more modules than you want to exclude. All modules not listed in EXCLUDE statements are copied. You can specify as many EXCLUDE statements as necessary.

If you follow a specification in the EXCLUDE statement with a colon (:), all modules beginning with the characters preceding the colon are excluded from the copying.

Note: you cannot use both the SELECT statement and the EXCLUDE statement in one PROC PDSCOPY step.

DETAILS

Output Data Set

The PDSCOPY procedure produces an output partitioned data set on disk or tape that contains copies of the requested members of the input partitioned data set.

Usage Note

If a module specified in a SELECT statement does not exist, PROC PDSCOPY issues a warning message and continues processing. This does not apply to

generic SELECT statements (that is, those containing a colon) or to EXCLUDE statements.

Printed Output

The PDSCOPY procedure prints:

1. INPUT and OUTPUT, the data set names and volume serials of the input and output libraries
2. MODULE, a list of the modules copied
3. the modules' ALIASes, if any
4. whether the copied modules REPLACED others of the same name
5. if a selected module or alias was NOT COPIED, a note explaining why.

If the output device is a disk, PROC PDSCOPY prints:

6. TRACKS, the sizes of the modules, in tenths of tracks
7. the SIZE, in decimal bytes, of the modules copied.

EXAMPLE

Copying Modules and Aliases Using PROC PDSCOPY

The example below copies all modules and aliases starting with the letters SAM. In this example, the alias must match the main module both by name and by TTR for the alias to be copied. Several of the modules replaced modules with the same name, and some modules were not copied because they did not match the TTR of the main module. (This occurs for obsolete alias names.)

```
// EXEC SAS
//OLD   DD  DSN=XXXXXX.SAMPLE.LOAD,DISP=SHR
//NEW   DD  DSN=XXXXXX.BACKUP.LOAD,DISP=OLD
//SYSIN DD  *
PROC PDSCOPY INDD=OLD OUTDD=NEW ALIASMATCH=BOTH SHAREINPUT;
   SELECT  SAM: ;
```

Output 41.1 PDSCOPY Example

```
1         SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST

NOTE: COPYRIGHT (C) 1985 SAS INSTITUTE INC., CARY, N.C.  27511, U.S.A.
NOTE: THE JOB SASTEST HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
      AT SAS INSTITUTE INC.       (XXXXXXXX).

NOTE: SAS OPTIONS SPECIFIED ARE:
      LINESIZE=120 SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.75 SECONDS.
1
2            PROC PDSCOPY INDD=OLD OUTDD=NEW ALIASMATCH=BOTH SHAREINPUT;
3             SELECT SAM: ;

   SAS PROC PDSCOPY VERSION 5.XX   01/16/85

❶  INPUT  DSNAME=XXXXXX.SAMPLE.LOAD,VOL=SER=QQQ333

   OUTPUT DSNAME=XXXXXX.BACKUP.LOAD,VOL=SER=RRR444

    ❷    MODULE ❻ TRACKS ❼ SIZE

         SAM32XX      2.4   35600 REPLACED
```

(continued on next page)

(continued from previous page)

```
          SAM3270      2.3   32976 REPLACED
          SAM3270P     2.3   33200 REPLACED
          SAM3279      2.3   32976 REPLACED
          SAM3287      2.2   31208 REPLACED
          SAM5150      2.1   24560 REPLACED
          SAM5150P     2.0   24560 REPLACED

            USED      78.6
            UNUSED     1.4
            TOTAL     80.0
            EXTENTS      4

          ALIAS     MODULE
          SAMXY749  SER281   NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAMXY750  SER281   NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32XXB  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32XXH  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32XXQ  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32XXV  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM3278   SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32782  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32783  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32784  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32792  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM32793  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM3287A  SAM3287  REPLACED
          SAM3287B  SAM3287  REPLACED
          SAM3287C  SAM3287  REPLACED
          SAM3287D  SAM3287  REPLACED
          SAM3287E  SAM3287  REPLACED
          SAM3287F  SAM3287  REPLACED
          SAM3290   SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).
          SAM3290B  SAM3279  NOT COPIED   (MAIN MODULE NOT PRESENT OR NOT MATCHED).

NOTE: THE PROCEDURE PDSCOPY USED 1.22 SECONDS AND 324K.
```

```
2         SAS(R) LOG   OS SAS 5.XX          MVS/XA JOB SASTEST  STEP SASTEST
NOTE: SAS USED 324K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Chapter 42

The PLOT Procedure

Operating systems: All

ABSTRACT

The PLOT procedure graphs one variable against another, producing a printer plot. The coordinates of each point on the plot correspond to the two variables' values in one or more observations of the input data set.

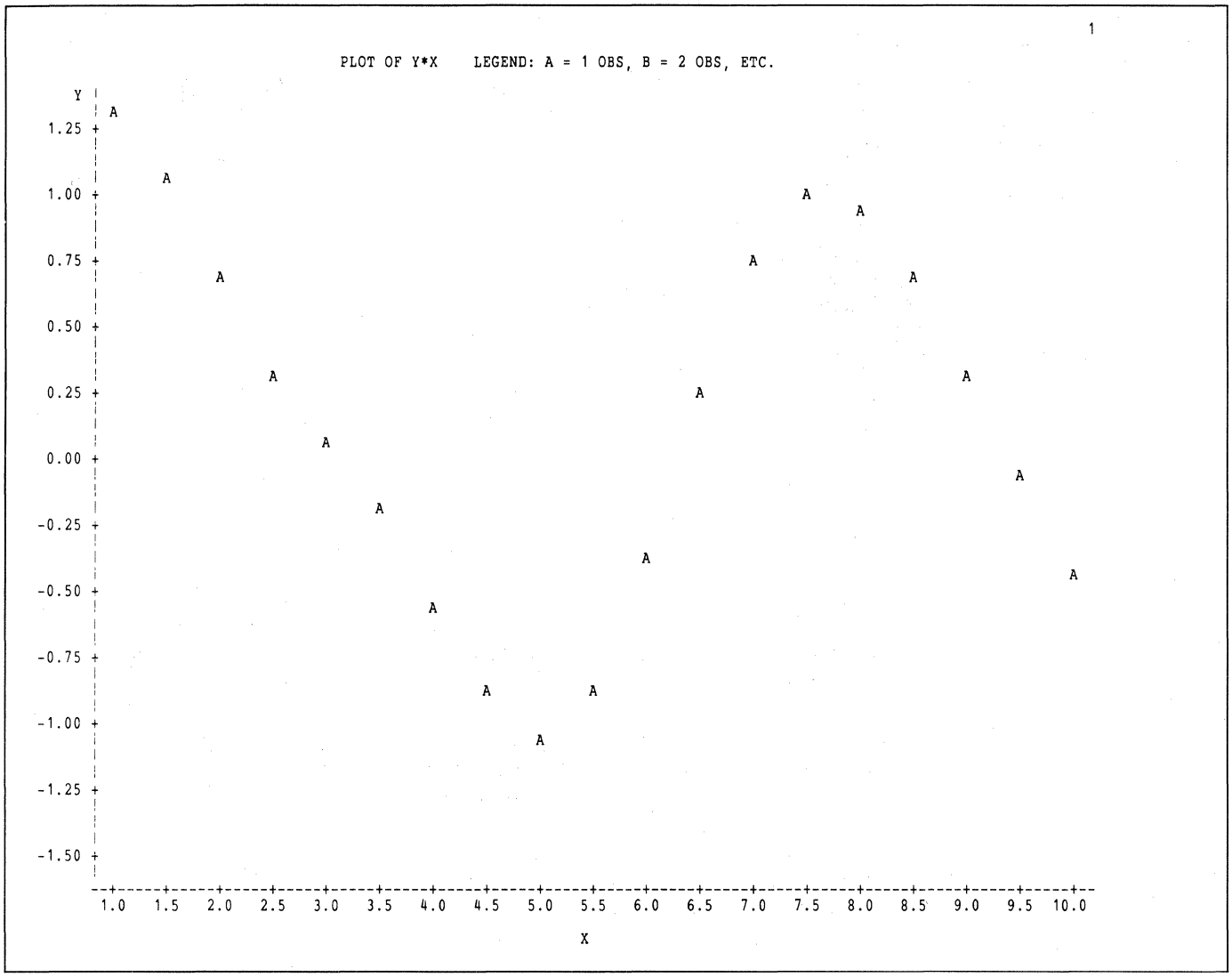
INTRODUCTION

The PLOT procedure takes the values that occur for each observation in an input SAS data set on two variables, say X and Y, and then represents the intersection of these values as points on the plot. All you need to do to produce a plot is to tell the procedure which variables to plot. For example, these statements

```
PROC PLOT;
   PLOT Y*X;
```

produce the simple plot shown in **Output 42.1**.

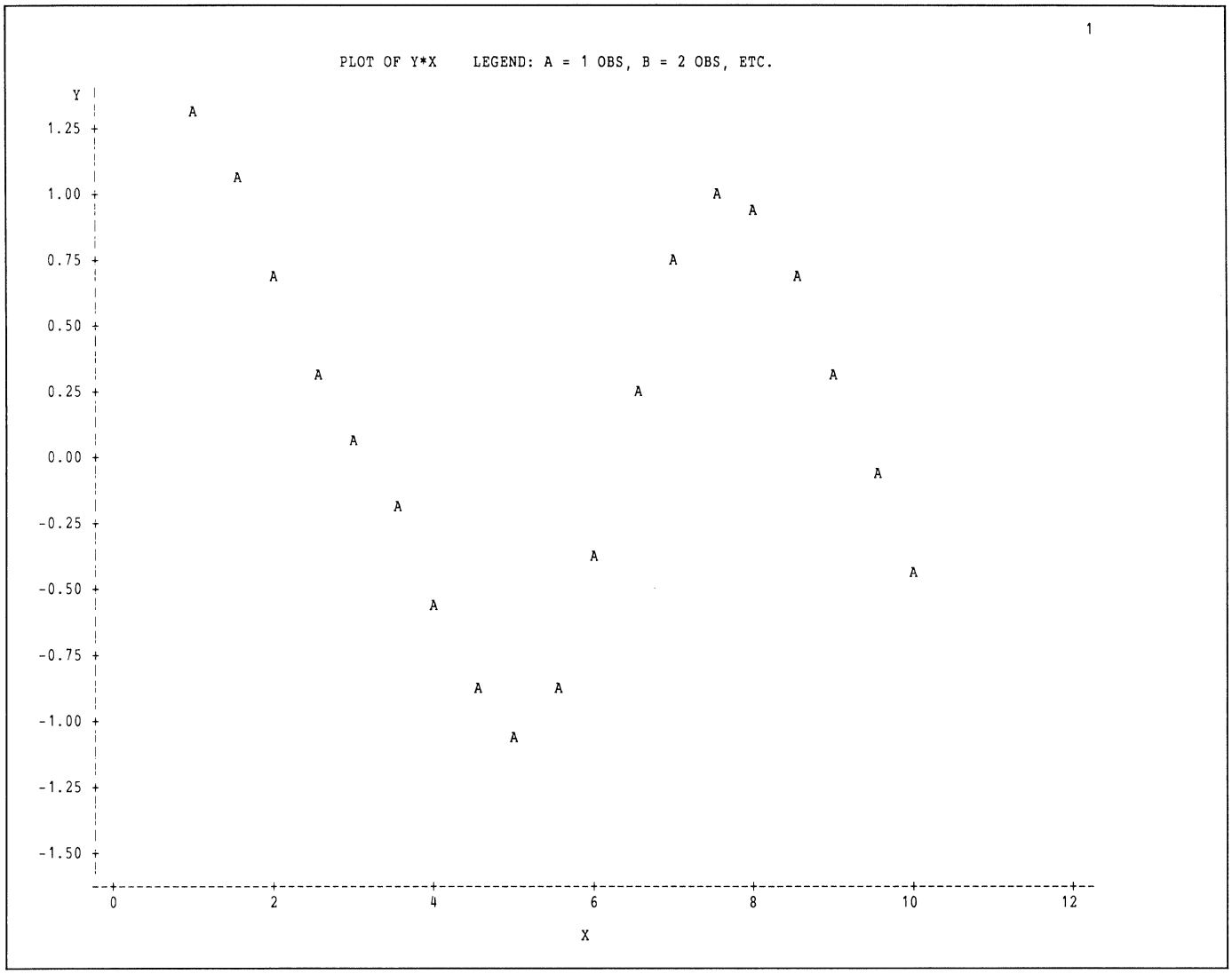
Output 42.1 Plotting Variables: PLOT Procedure



Note that PROC PLOT automatically scales the values of X and Y on the plot. Or you can specify the tick marks, the lines on the axis marking specific values that you want. For example,

```
PROC PLOT;
   PLOT Y*X  / HAXIS=0 TO 12 BY 2;
```

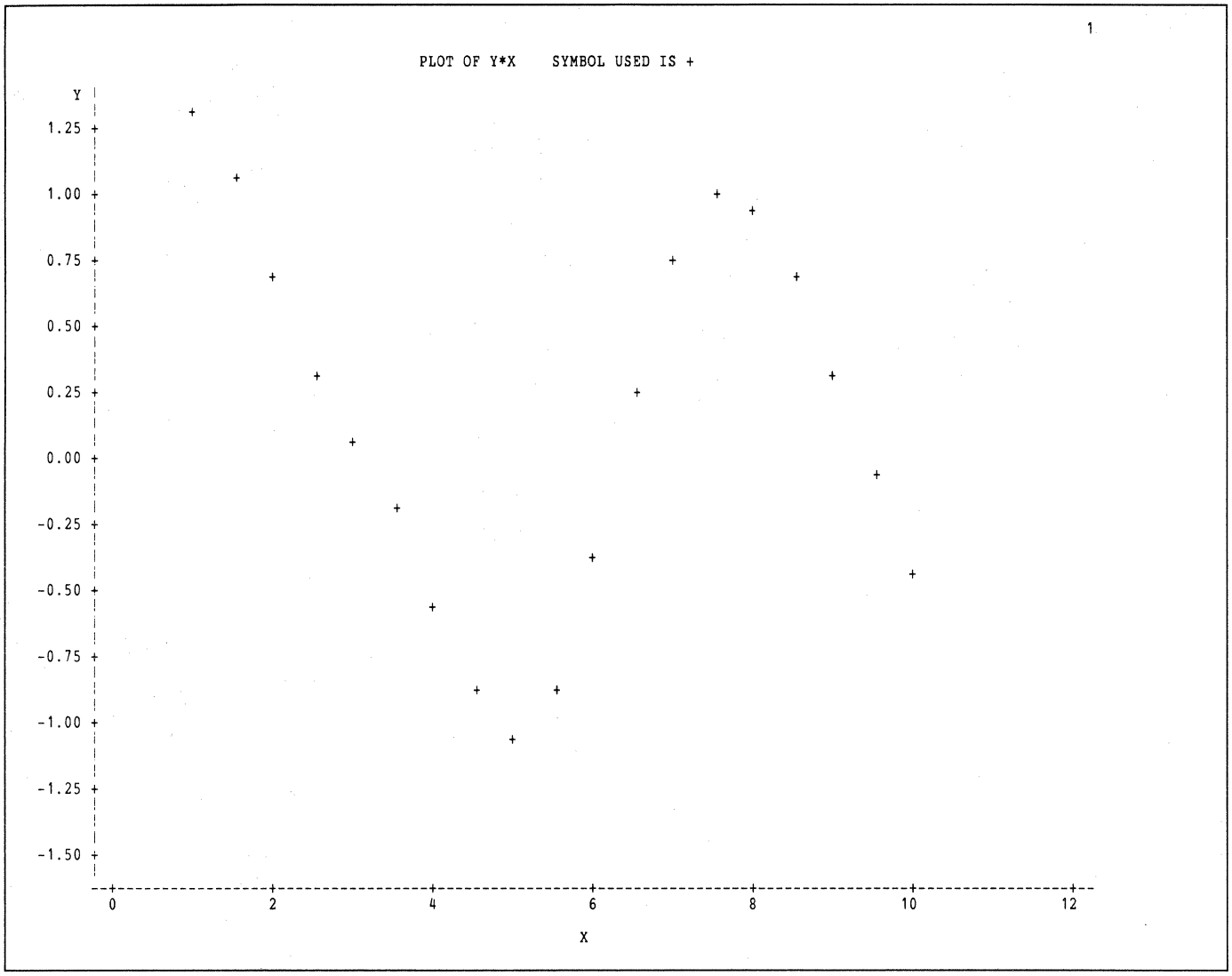
Output 42.2 Specifying Tick Marks: PLOT Procedure



In this example, PROC PLOT uses the plotting character A to indicate that one observation occurs at each point, B to indicate two observations, and so on. **Output 42.3** illustrates how you can specify another symbol, like a plus sign (+), to be used as the plotting character.

```
PROC PLOT;
   PLOT Y*X='+' / HAXIS=0 TO 12 BY 2;
```

Output 42.3 Specifying Other Plotting Symbols: PLOT Procedure

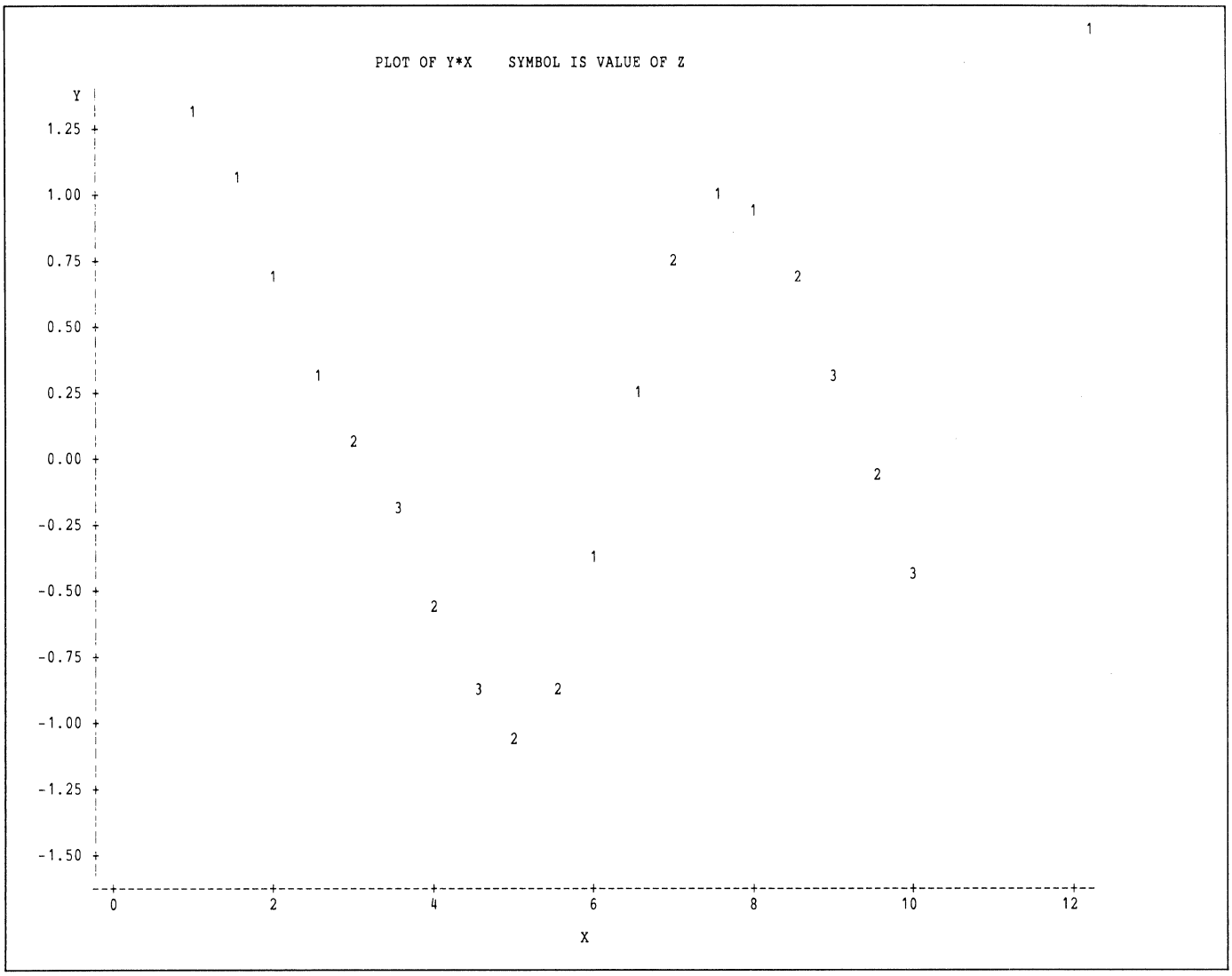


Output 42.4 shows how you can use the values of a third variable to determine the plotting character. For example, suppose another variable Z takes on the values 1, 2, and 3. These statements

```
PROC PLOT;
   PLOT Y*X=Z / HAXIS=0 TO 12 BY 2;
```

indicate that the plotting character used for a particular observation is the value of Z for that observation.

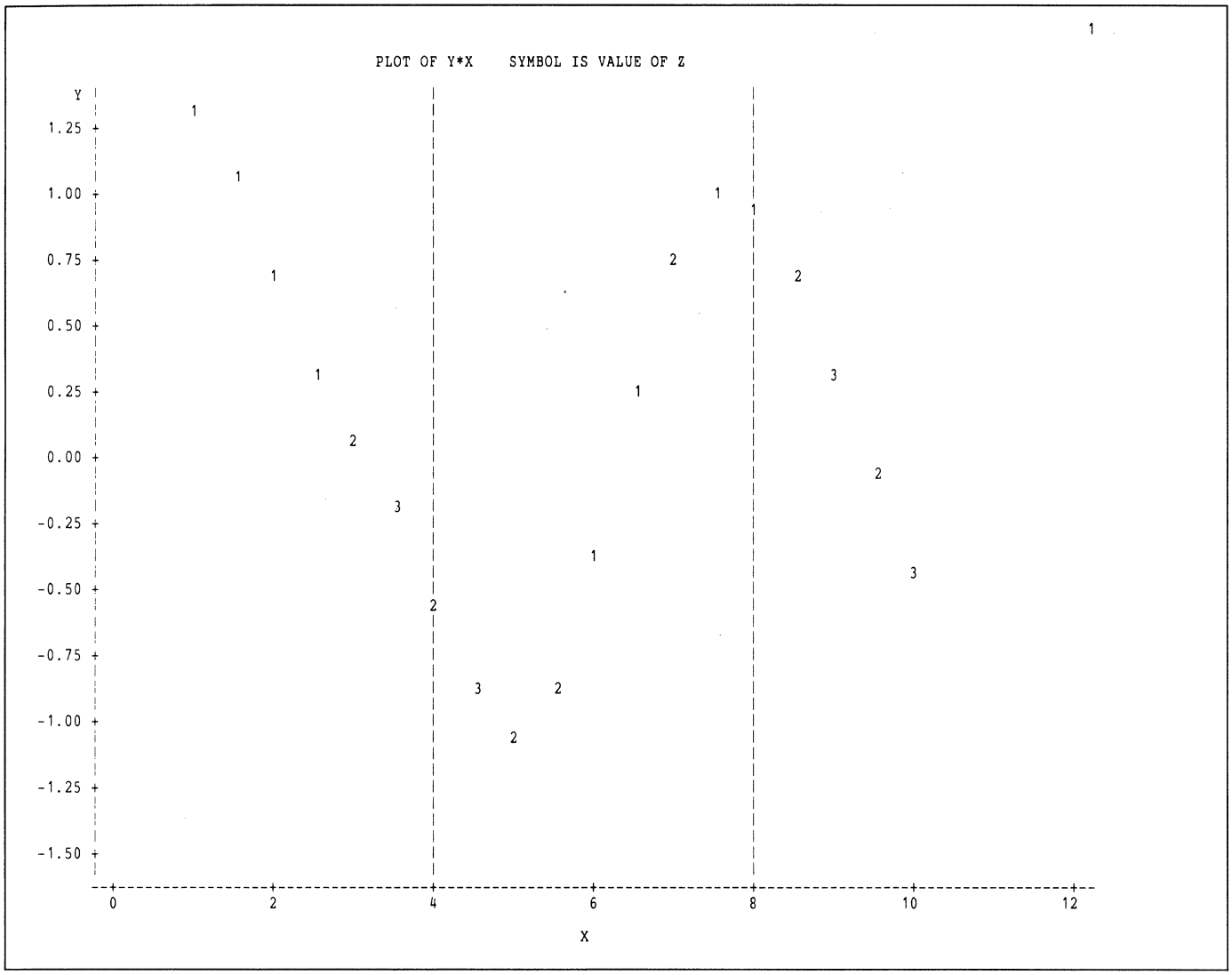
Output 42.4 Using the Values of a Third Variable to Determine Plotting Characters: PLOT Procedure



Another way to enhance the information contained in the plot is to use reference lines at specific values. For example, for reference lines perpendicular to the horizontal axis, use these statements:

```
PROC PLOT;
   PLOT Y*X=Z / HAXIS=0 TO 12 BY 2 HREF=4 8;
```

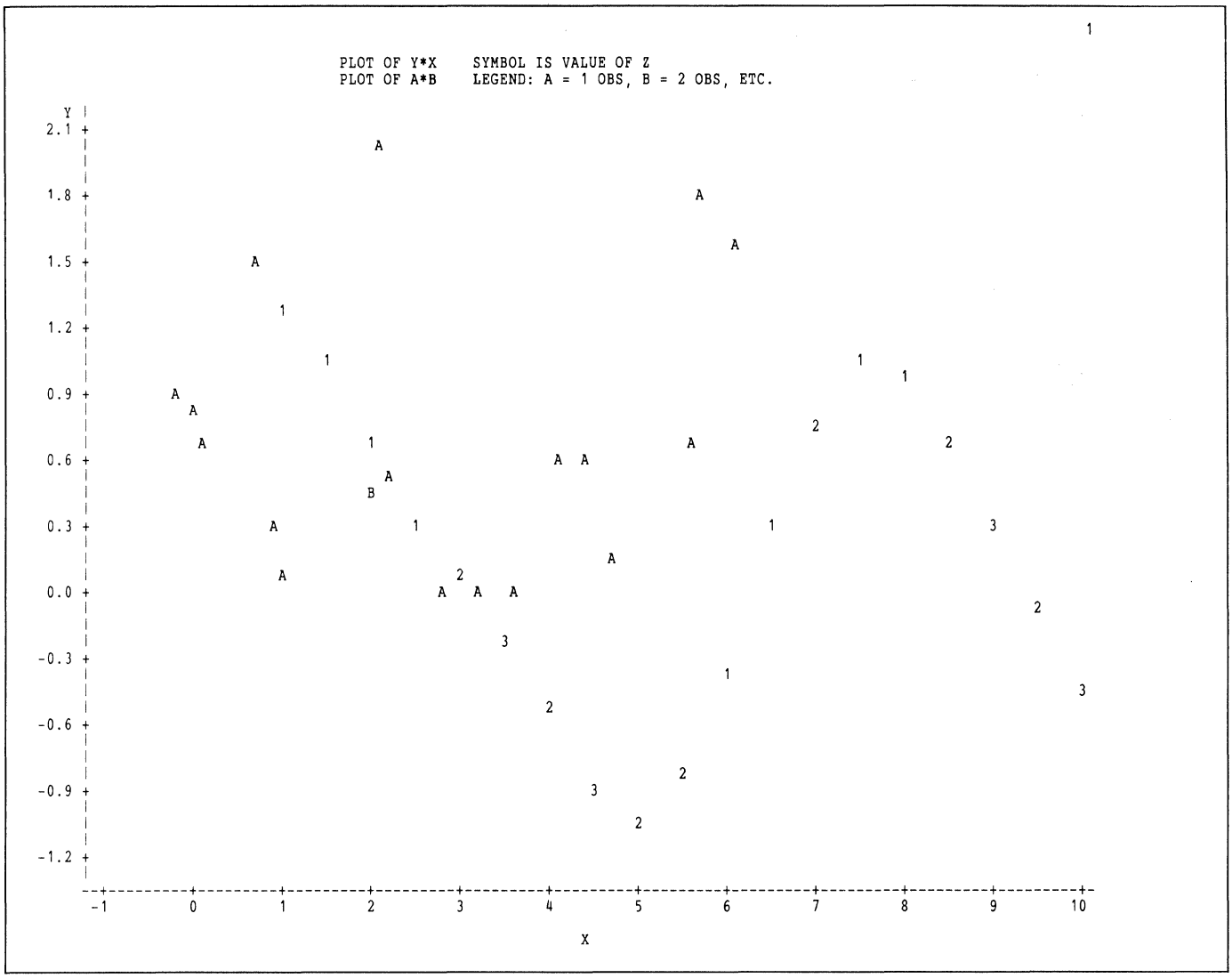
Output 42.5 Specifying Reference Lines: PLOT Procedure



Output 42.6 shows that you can also superimpose two or more plots on the same axes.

```
PROC PLOT;
   PLOT Y*X=Z A*B / OVERLAY;
```

Output 42.6 Superimposing Two or More Plots on the Same Axes: PLOT Procedure



You can also use PLOT to:

- plot character as well as numeric variables
- specify the length and width of the plot
- reverse the order of the values on the vertical axis
- draw contour plots with shading intensity determined by a third variable in the data set.

The SAS statements that generated the data for the plots shown in this section are listed at the end of the chapter in **Examples of PROC PLOT Output**.

SPECIFICATIONS

The PLOT procedure is controlled by these statements:

PROC PLOT *options*;

BY *variables*;
PLOT *requests / options*;

A PLOT statement must be present to tell the procedure which variables to plot. Any number of PLOT statements can be specified each time the procedure is invoked, and any number of plots can be requested on one PLOT statement.

PROC PLOT Statement

PROC PLOT *options*;

The options below can appear in the PROC PLOT statement.

DATA=*SASdataset*
names the input SAS data set to be used by PROC PLOT. When the DATA= option is omitted, PLOT uses the most recently created SAS data set.

UNIFORM
requests uniform axis scaling across BY groups when a BY statement is used. The same scaling allows you to compare the plots for different levels of the BY variables directly.

NOLEGEND
suppresses printing the legend at the top of each plot. The legend lists the names of the variables being plotted and the plotting characters used in the plot.

BY Statement

BY *variables*;

A BY statement can be used with PROC PLOT to obtain separate plots on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. PLOT produces a new page each time a new value of the BY variable is encountered in the data set. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

PLOT Statement

PLOT *requests / options*;

The PLOT statement requests the plots to be produced by PROC PLOT. Remember that you can include many PLOT statements and specify many plot requests on one PLOT statement.

These terms can be used in the PLOT statement:

request
specifies the variables (vertical and horizontal) to be plotted and the plotting character to be used to mark the points on the plot. The request can take these forms:

*vertical*horizontal*
*vertical*horizontal='character'*
*vertical*horizontal=variable*

*vertical*horizontal*

names the variable to be shown on the vertical axis and the variable to be shown on the horizontal axis.

For example, the statements

```
PROC PLOT;
   PLOT Y*X;
```

request a plot of Y by X. Y appears on the vertical axis, X on the horizontal.

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 twenty-six or more observations at the same printing position.

*vertical*horizontal='character'*

names the variables to be shown on the vertical and horizontal axes and also specifies a character constant to mark each point on the plot. A single character is used to represent values from one or more observations.

For example, the statements

```
PROC PLOT;
   PLOT Y*X='+';
```

request a plot of Y by X, with each point on the plot represented by a plus sign (+).

*vertical*horizontal=variable*

names the variables to be plotted on the vertical and horizontal axes and also 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. When more than one observation maps to the same plotting position, the value from the first observation marks the point. For example,

```
PROC PLOT;
   PLOT HEIGHT*WEIGHT=SEX;
```

SEX is a character variable with values of FEMALE and MALE: the values F and M mark each observation on the plot.

Note: the plotting symbol is the first nonblank character of each value even if more than one value starts with the same letter.

To request two or more plots, write one request after another:

```
PROC PLOT;
   PLOT A*B R*S;
```

If you want to plot all the combinations of one set of variables with another, you can use a grouping specification. Enclose each set of variables in parentheses, joining them with an asterisk (*). For example, the following PLOT statements are equivalent:

```
PLOT (Y X)*(A B);

PLOT Y*A Y*B X*A X*B;
```

You can also abbreviate a variable list to request a number of plots:

```
PLOT Y*(A--Z);
```

The options below can appear in the PLOT statement. A slash (/) separates these options from the plot requests. No slash is needed if no options are specified.

Scale of axes

VAXIS=*values* specifies the tick mark values to be equally spaced along the vertical axis. When the variable is numeric, VAXIS= values must be given in either ascending or descending order.

The statements

```
PROC PLOT;
   PLOT Y*X / VAXIS=10 TO 100 BY 5;
```

ask for a plot of Y by X, with tick marks at 10, 15, 20 and on up to 100 on the vertical axis.

Numeric values need not be uniformly distributed; a specification of the form

```
VAXIS=10 100 1000 10000
```

is valid and produces a logarithmic plot.

For character variables, the values can be given in any order.

In addition, the following VAXIS= specifications are valid

```
VAXIS='01JAN85'D TO '01JAN86'D BY MONTH
```

or

```
VAXIS='01JAN85'D TO '01JAN86'D BY QTR
```

In these examples, the FROM and TO values can be any of the valid SAS date, time, or datetime values described for the SAS functions INTCK and INTNX (see the chapter "SAS Functions"). The BY value can be any of the valid values listed for the *interval* argument in the SAS functions INTCK and INTNX. You must use a FORMAT statement to print the tick mark values in an understandable form.

HAXIS=*values* specifies tick mark values for the horizontal axis. The HAXIS= option follows the same rules as the VAXIS= option above.

VZERO requests that tick marks on the vertical axis begin in the first position with a zero. The VZERO request is ignored if the vertical variable has negative values or if an optional VAXIS= specification does not begin with zero.

HZERO requests that tick marks on the horizontal axis begin in the first position with a value of zero. The HZERO request is ignored if the horizontal variable has negative values or if an optional HAXIS= specification does not begin with zero.

VREVERSE asks that the order of the values on the vertical axis be reversed. (No option is available to reverse the horizontal axis.)

Reference lines

VREF=*values* requests that a horizontal line be drawn on the plot at the specified values on the vertical axis.

VREFCHAR='*c*' gives the character to define the horizontal lines requested by the VREF= option. If you do not specify a character with VREFCHAR=, the hyphen (-) is used.

HREF=*values* requests that a vertical line be drawn on the plot at the specified values on the horizontal axis. For example, the statements

```
PROC PLOT;
   PLOT Y*X / HREF=5;
```

request a plot of Y by X with a vertical line at 5 on the horizontal axis.

HREFCHAR='*c*' specifies the character to define the vertical lines requested by the HREF= option. If you do not use HREFCHAR=, the vertical bar (|) is used.

Plot size Plots normally occupy one page each. You can use the options below to change the length and width of the plot. You can also use the system options LINESIZE= and PAGESIZE= to change plot sizes. (See the OPTIONS statement.)

VPOS=*n* specifies the number of vertical print positions on the vertical axis. The maximum VPOS= must be at least eight less than the page size; the exact number depends on the titles used, whether plots are overlaid, and whether CONTOUR is specified.

HPOS=*n* specifies the number of print positions on the horizontal axis. At least three additional positions are needed to print vertical axis information; the exact number depends on the number of characters in the vertical variable's values.

VSPACE=*n* specifies the number of print lines to be used between tick marks on the vertical axis.

HSPACE=*n* specifies the number of print positions to be used between tick marks on the horizontal axis.

Overlaying plots

OVERLAY requests PROC PLOT to combine the plots specified in the PLOT statement on one page. The variables in the first plot label the axes. Unless HAXIS= or VAXIS= is given, the axes are automatically scaled in the best way to fit all the variables, and the variable labels (if any)

associated with the first pair of variables are printed next to the axes. When the SAS system option OVP is in effect and overprinting is allowed, the plots are superimposed; otherwise, when NOOVP is in effect, points appearing in more than one plot are represented by the value from the first plot. Using the OVP system option and the OVERLAY option is the only case where the PLOT procedure overprints.

Contour plots

CONTOUR=*value* requests plotting symbols with varying degrees of shading where *value* is the number of levels for dividing the range of the response 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. CONTOUR's value can range from one to ten. For example, the statements

```
PROC PLOT;
   PLOT A*B=Z / CONTOUR=10;
```

request a plot whose points vary in darkness depending on the value of Z. Since the CONTOUR= value is ten, ten darkness levels are used.

Overprinting is used to produce the shading if it is allowed. Otherwise, single characters varying in darkness are used. The CONTOUR= option is most effective when the plot is dense.

S1='*value*'
S2='*value*'
...
specify plotting symbols to use for each contour value in ascending order. When PROC PLOT produces contour plots, it automatically chooses the symbols to use for each level of intensity. It is possible to override these symbols and specify your own symbols for intensity levels using the S options. Up to three characters can be specified in quotes. If overprinting is not allowed, only the first character is used. For example, to specify three levels of shading for the Z variable, these statements may be used:

```
PROC PLOT;
   PLOT X*Y=Z / CONTOUR=3
        S1='A' S2='+' S3='X0A';
```

The symbols can be specified as hex constants.

```
PROC PLOT;
   PLOT X*Y=Z / CONTOUR=3
        S1='7A'X S2='7A'X S2='7F'X S3='A6'X;
```

This feature was designed especially for IBM 3800 printers where the hex constants can represent grey-scale fill characters.

DETAILS

Missing Values

If values on either of the plotting variables are missing, PROC PLOT does not include the observation in the plot. However, in a plot of X*Y, values of X with corresponding missing values of Y are included in scaling the X axis.

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.

Since PROC PLOT prints one character for each observation, using SAS program statements to generate the data set for PLOT can enhance the effectiveness of continuous plots (see **Example 3**) and contour plots (see **Example 5**).

For example, suppose that you want to generate data in order to plot the equation

$$y = 2.54 + 3.83 * x$$

for x ranging from 0 to 100 with these statements:

```
OPTIONS LINESIZE=80;
DATA GENERATE;
   DO X=0 TO 100 BY 2;
      Y=2.54+3.83*X;
      OUTPUT;
      END;

PROC PLOT;
   PLOT Y*X;
```

If the plot is printed with a LINESIZE= value of eighty, about seventy-five positions are available on the horizontal axis for the X values. Thus, two is a good increment: fifty-one observations are generated, which is less than seventy-five.

However, if the plot is printed with a LINESIZE= value of 132, an increment of two produces a plot with a space between each plotting character. For a smoother line, a better increment is one, since 101 observations are generated.

Printed Output

Each plot uses one full page unless the plot's size is changed by the VPOS= and HPOS= options or the system options PAGESIZE= and LINESIZE=. Titles, legends, and variable labels are printed at the top of each page. Each axis is labeled with the variable name or the variable label, if one is specified. (See the LABEL statement in "Statements Used in the PROC Step.")

EXAMPLES

Plotting Observed Data: Example 1

This example plots the two variables RATED and ACTUAL against each other. PROC PLOT uses an increment of five for tick marks on the vertical axis; tick marks on the horizontal axis have an increment of four. Since two observations have a RATED value of 118 and an ACTUAL value of 121, PROC PLOT uses a B to represent the point corresponding to RATED=118 and ACTUAL=121. Other points are represented by As.

```
DATA SPEED;
   INPUT RATED ACTUAL;
   LABEL RATED='RATED SPEED'  ACTUAL='ACTUAL SPEED';
   CARDS;
 75  85
110 112
 75  81
105 108
112 115
 75  77
 90  89
 70  73
118 121
103 100
118 121
;
PROC PLOT;
   PLOT RATED*ACTUAL;
   TITLE 'RATED SPEED VS. ACTUAL SPEED';
```

See **Output 42.7**.

Using a Plot Character and Defining Tick Marks: Example 2

This example uses variables from an analysis of covariance: RESPONSE is the response variable, and NUISANCE is a presumed covariate. To keep track of which observations are associated with different values of the variable TREAT, values of TREAT are used for the plotting characters.

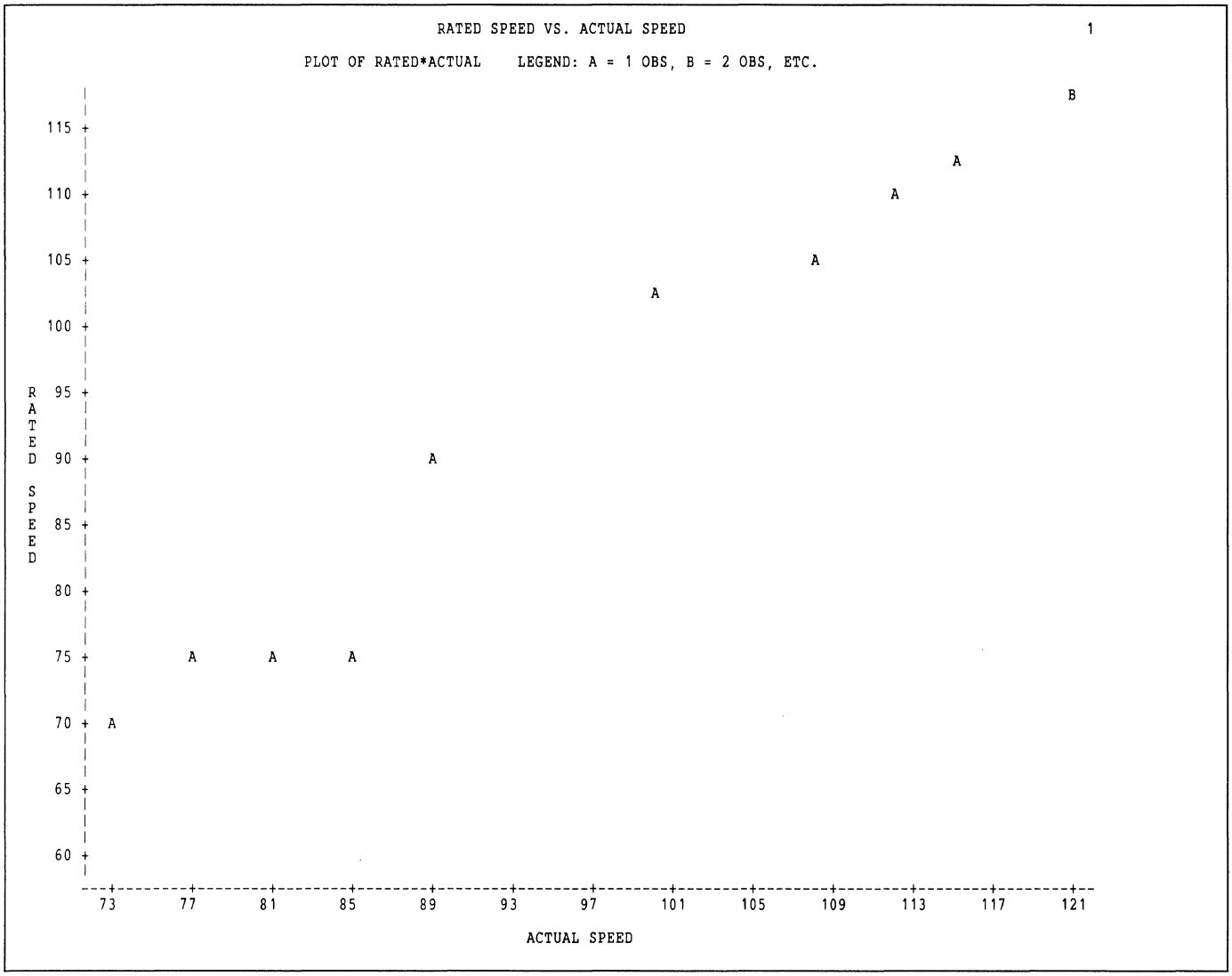
You can use the VAXIS= option to define tick marks for the vertical axis, beginning with 125 and going up to 185 by 5s.

```
DATA COVAR;
   INPUT TREAT $ RESPONSE NUISANCE;
   CARDS;
P 125 3.1
P 135 9.0
P 144 14.9
P 153 20.2
Q 136 2.0
Q 152 8.5
Q 160 12.6
Q 165 17.1
R 154 3.2
R 164 10.5
R 173 15.4
```

```
R 183 20.7
PROC PLOT;
   PLOT RESPONSE*NUISANCE=TREAT / VAXIS=125 TO 185 BY 5;
   TITLE 'RESPONSE VS. NUISANCE';
   TITLE2 'THREE TREATMENTS';
```

See **Output 42.8.**

Output 42.7 Plotting Observed Data: PLOT Procedure



Plotting the Graph of an Equation: Example 3

In this example, program statements in the DATA step are used to create 115 observations to plot the equation

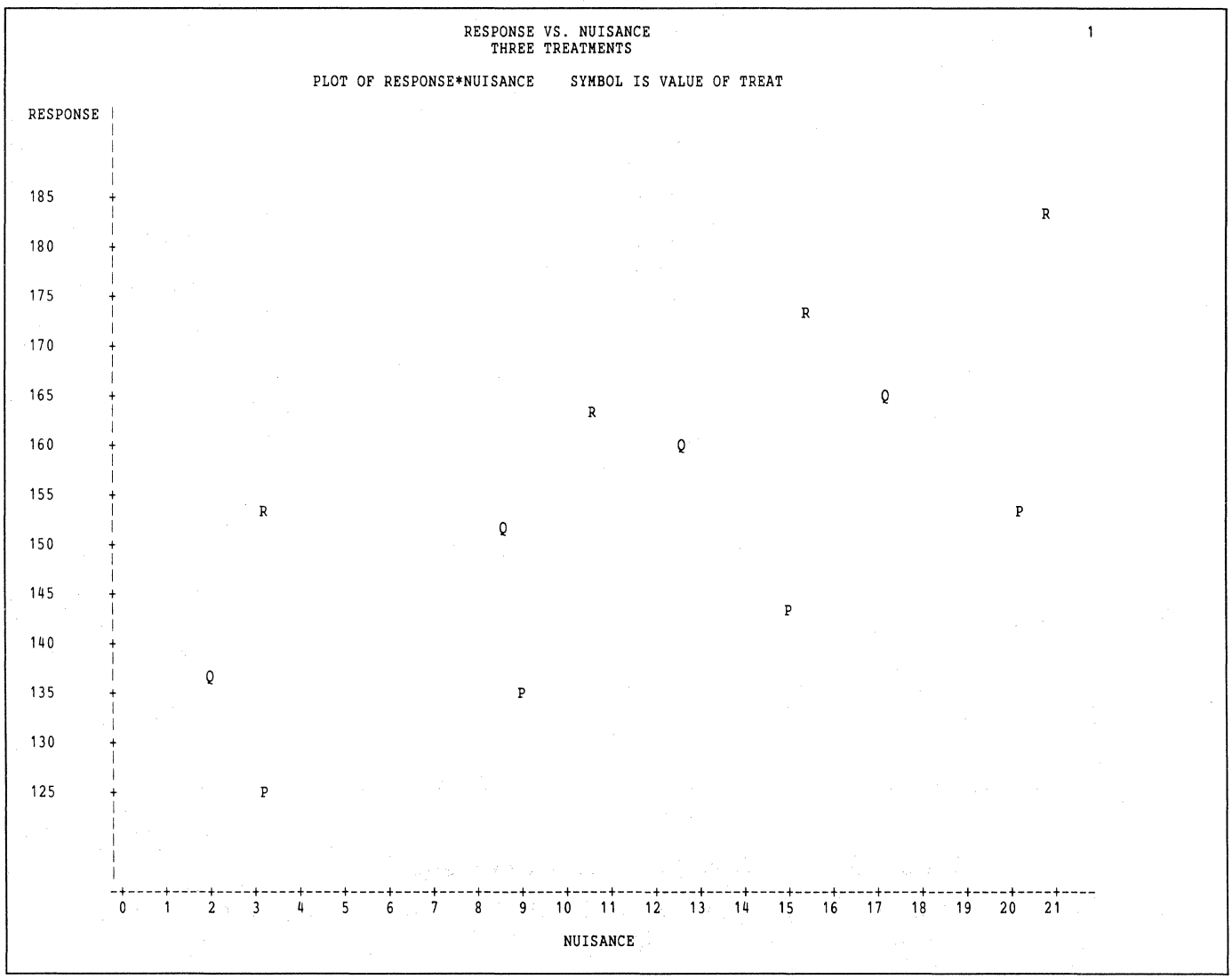
$$Y = X \sin(2X)$$

for values of X ranging from zero to five.

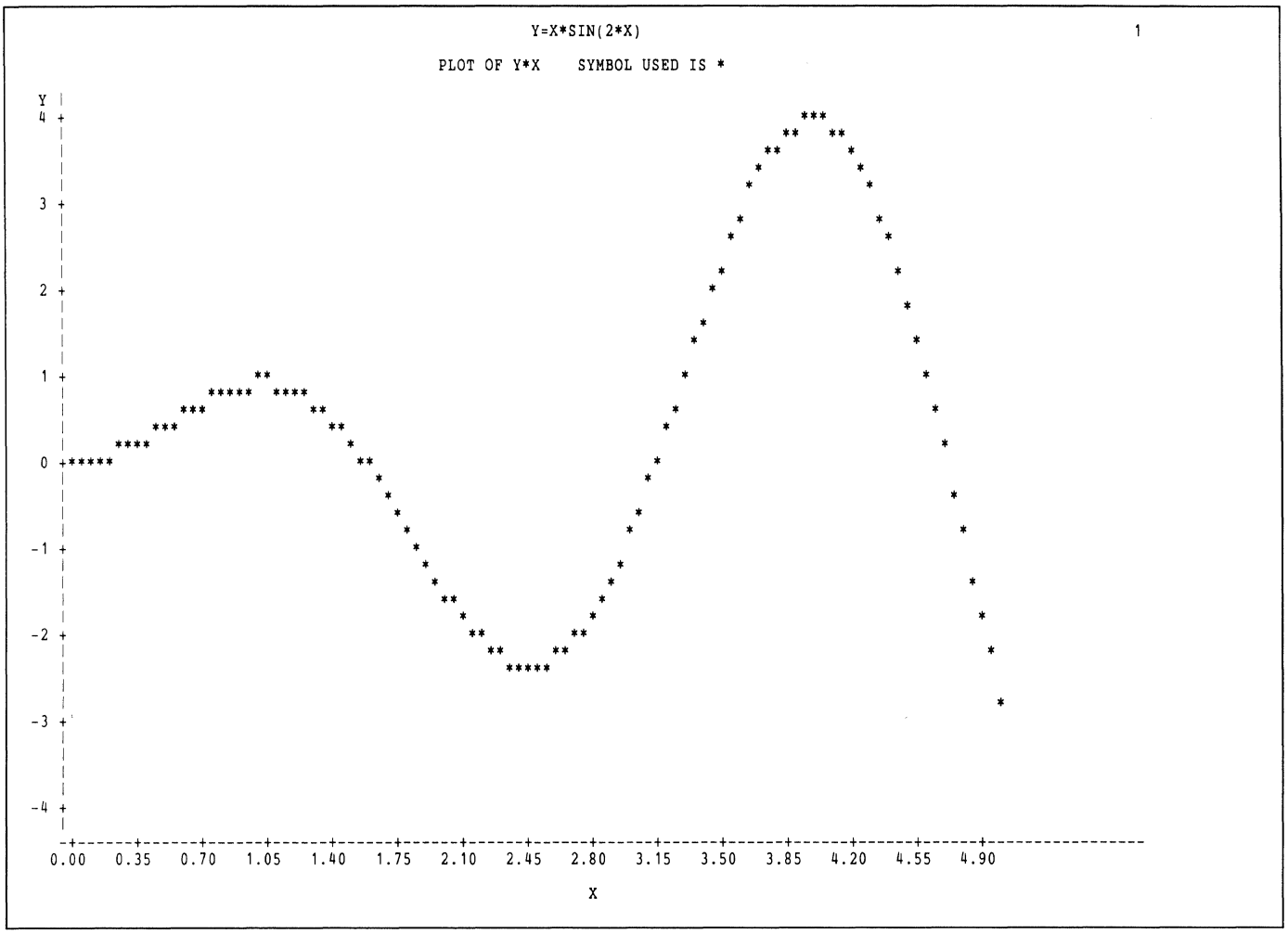
No INPUT statement is used; all the X and Y values are generated by the program statements, and the OUTPUT statement creates the observations. In the PLOT statement, an asterisk (*) is specified as the plotting character.

```
DATA PROGRAM;
   DO X=0 TO 5 BY .05;
      Y=X*SIN(2*X);
      OUTPUT;
      END;

PROC PLOT;
   PLOT Y*X='*';
   TITLE 'Y=X*SIN(2*X)';
```

Output 42.8 Using a Plot Character and Defining Tick Marks: PLOT Procedure



Output 42.9 Plotting the Graph of an Equation: PLOT Procedure



Plotting Predicted vs. Actual Values: Example 4

It is often useful to plot the graph of an equation and the observed data for which the equation was developed.

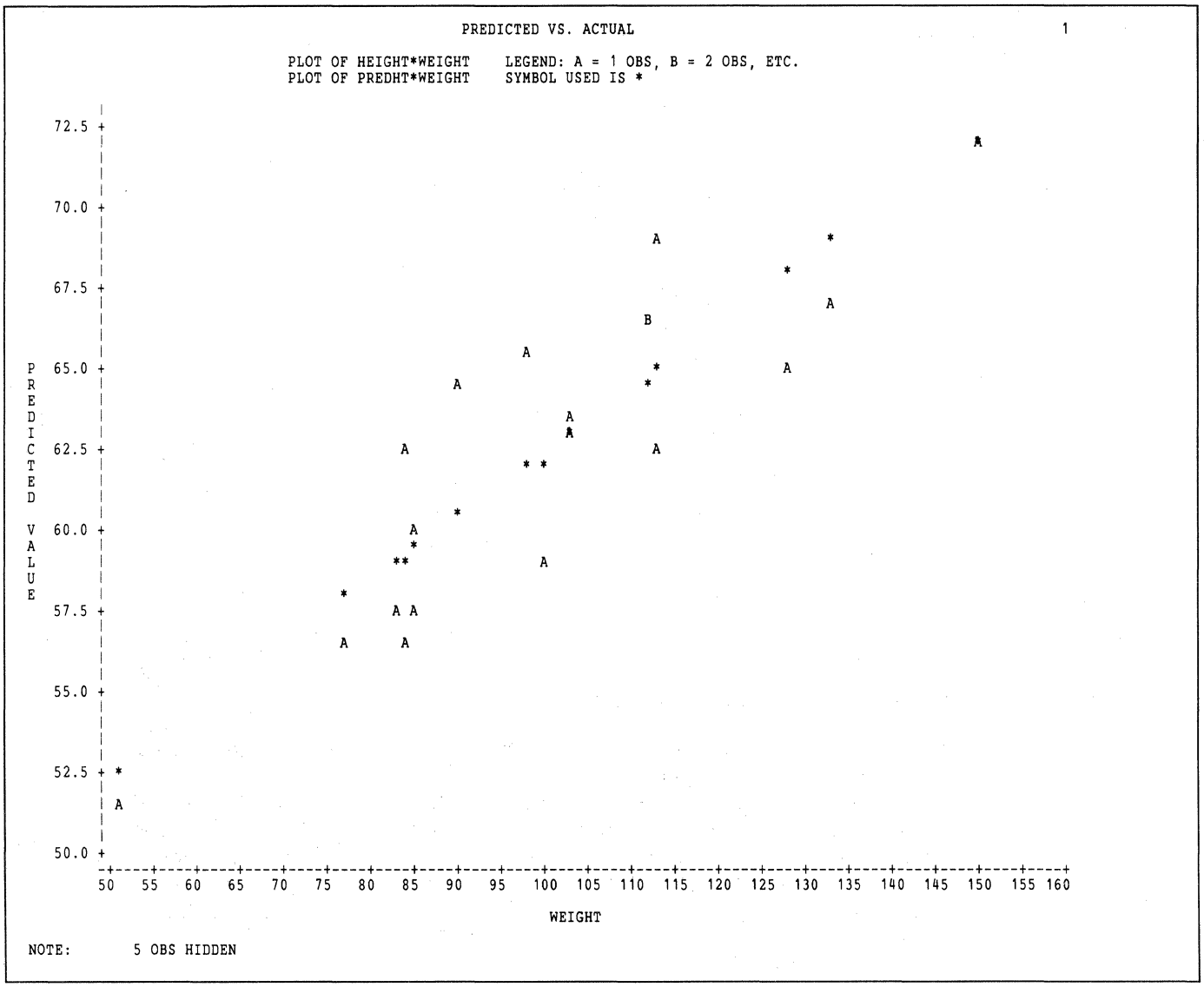
In regression analysis you often want to plot the regression equation as well as the actual values. You can use the OUTPUT statement with PROC REG to produce a data set containing both observed and predicted values, and then use PROC PLOT to overlay plots of these values. Here is an example:

```
DATA HTWT;
   INPUT HEIGHT WEIGHT @@;
   CARDS;
69.0 112.5 56.5 84.0 65.3 98.0 62.8 102.5 63.5 102.5
57.3 83.0 59.8 84.5 62.5 112.5 62.5 84.0 59.0 99.5
51.3 50.5 64.3 90.0 56.3 77.0 66.5 112.0 72.0 150.0
64.8 128.0 67.0 133.0 57.5 85.0 66.5 112.0
;
PROC REG;
   MODEL HEIGHT=WEIGHT;
   OUTPUT OUT=BOTH P=PREDHT;
```

```
PROC PLOT DATA=BOTH;
   PLOT HEIGHT*WEIGHT PREDHT*WEIGHT='*' / OVERLAY;
   TITLE 'PREDICTED VS. ACTUAL';
```

Actual values are indicated with As, predicted values with asterisks (*).
See **Output 42.10**.

Output 42.10 Plotting Predicted vs. Actual Values: PLOT Procedure



Contour Plotting: Example 5

It is possible to represent the values of three variables using a two-dimensional plot by setting one of the variables as the CONTOUR variable. When the value of that variable is high, it is represented by a dark point on the plot; when the value is low, it is represented by a light point.

In the example below, the three variables represented in the plot are X, Y, and Z. X and Y appear on the axes, and Z is the contour variable. You can see from the plot where high values of Z occur: if this were a three-dimensional plot, the dark areas would be a hilltop. Program statements are used to generate the observations for the plot, and the equation

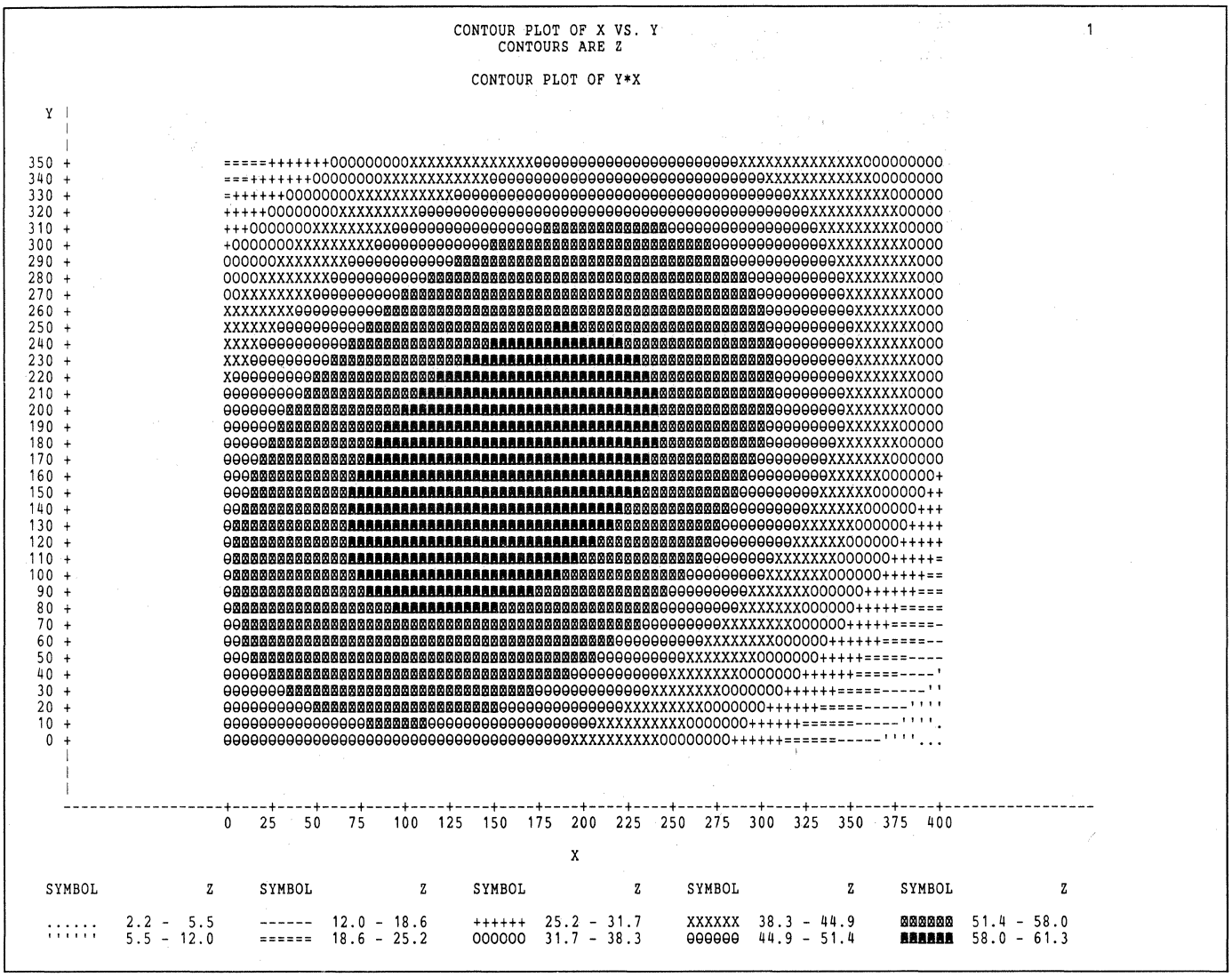
$$Z=46.2 + .09X-.0005X^2 + .1Y-.0005Y^2+ .0004XY$$

describes the contour surface. Data set CONTOURS contains observations with values of X ranging from 0 to 400 by 5 and for values of Y ranging from 0 to 350 by 10.

The shadings associated with the values of Z appear at the bottom of the plot.

```
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;

PROC PLOT;
   PLOT Y*X=Z / CONTOUR=10;
   TITLE 'CONTOUR PLOT OF X VS. Y';
   TITLE2 'CONTOURS ARE Z';
```

Output 42.11 Contour Plotting: PLOT Procedure



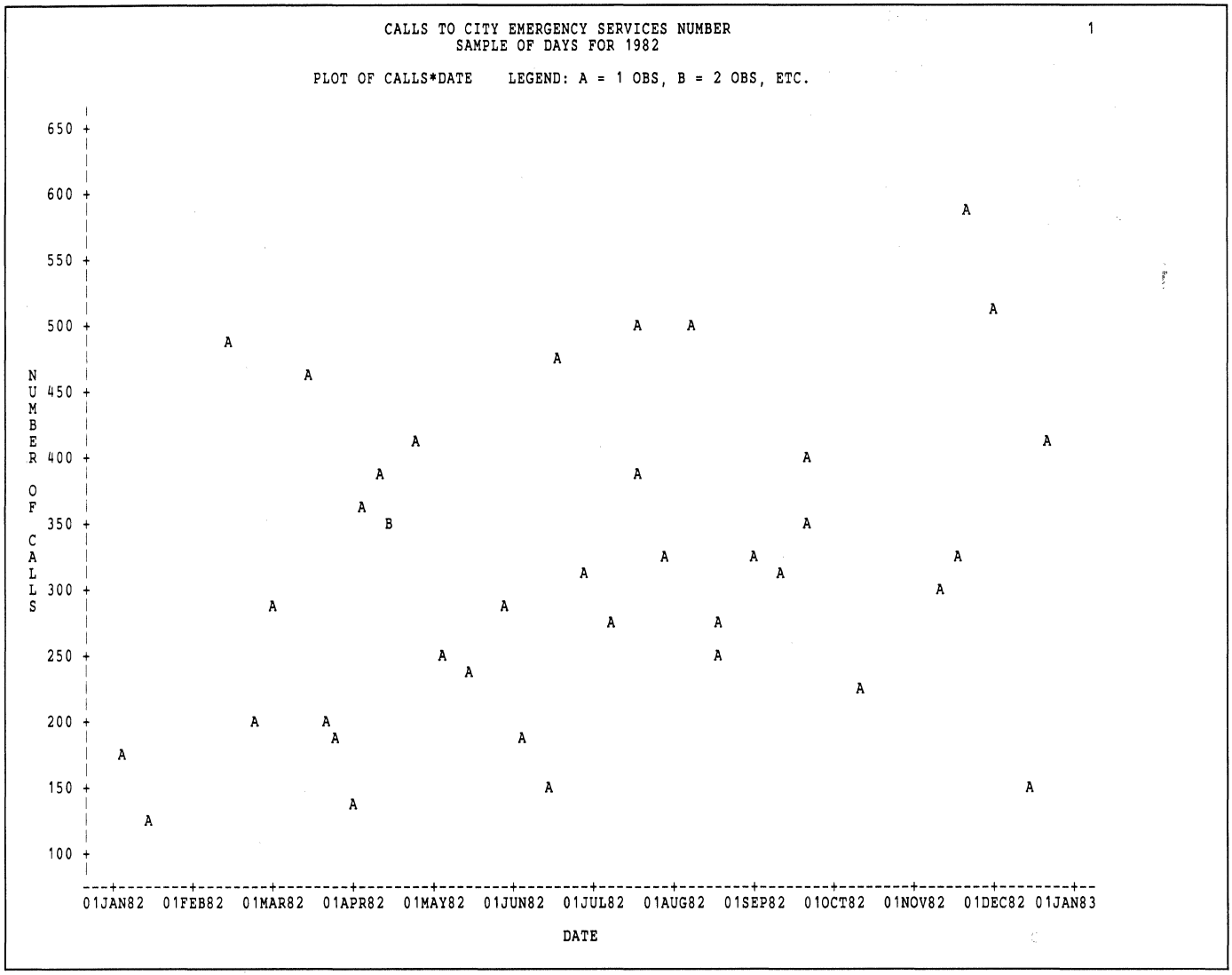
Labeling an Axis with Date Values: Example 6

When you plot a variable against a date variable, you can label the tick marks on an axis with date values by using SAS date constants after the VAXIS= or HAXIS= option. (See "SAS Expressions" for a description of date, time, and date-time constants.) To cause the constants to be printed in an understandable form, specify a format for the date variable. This example uses date constants to label the tick marks on the horizontal axis with the first day of each month. Note that in order to have room on the axis for observations in the month of December, you must specify a tick mark for January of the following year.

```
DATA SAMPLE;
  INPUT DATE:DATE7. CALLS;
  LABEL DATE='DATE'
        CALLS='NUMBER OF CALLS';
  CARDS;
1APR82 134
2MAR82 289
```

```
3JUN82 184
4JAN82 179
5APR82 360
6MAY82 245
7JUL82 280
8AUG82 494
9SEP82 309
11APR82 384
21MAR82 201
13JUN82 152
14JAN82 128
15APR82 350
15DEC82 150
16MAY82 240
17JUL82 499
18AUG82 248
19SEP82 356
10OCT82 222
11NOV82  294
2DEC82  511
22DEC82 413
13FEB82  488
14MAR82  460
15APR82  356
16JUN82  480
17JUL82  388
17NOV82  328
18AUG82  280
19SEP82  394
23NOV82  590
24FEB82 201
25MAR82 183
26APR82 412
27MAY82 292
28JUN82 309
29JUL82 330
30AUG82 321
;

PROC PLOT;
   PLOT CALLS*DATE/HAXIS='1JAN82'D '1FEB82'D '1MAR82'D '1APR82'D
   '1MAY82'D '1JUN82'D '1JUL82'D '1AUG82'D '1SEP82'D '1OCT82'D
   '1NOV82'D '1DEC82'D '1JAN83'D;
   FORMAT DATE DATE7.;
   TITLE 'CALLS TO CITY EMERGENCY SERVICES NUMBER';
   TITLE2 'SAMPLE OF DAYS FOR 1982';
```

Output 42.12 Labeling an Axis with Data Values: PLOT Procedure



Examples of PROC PLOT Output

The data for the examples in the earlier section of this chapter were generated with the following SAS statements.

```
DATA GENERATE;
   DO X=1 TO 10 BY .5;
      Y=SIN(X) + SIN(2*X)/(X+1);
      Z=CEIL(UNIFORM(13131)*3);
      A=2*MOD(X,Y);
      B=3*(1-MOD(Y,X));
      OUTPUT;
      END;
```

Chapter 43

The PRINT Procedure

Operating systems: All

ABSTRACT

The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. Totals and subtotals for numeric variables can also be printed.

INTRODUCTION

PROC PRINT prints a listing of the values of some or all of the variables in a SAS data set. You can produce customized reports using procedure options and statements; for example, with a BY statement, PROC PRINT separates observations into groups defined by the BY variables. Numeric variable totals are printed if a SUM statement is used.

SPECIFICATIONS

The following statements can be used with PROC PRINT:

PROC PRINT *options*;
  VAR *variables*;

ID *variables*;
BY *variables*;
PAGEBY *byvariable*;
SUM *variables*;
SUMBY *byvariable*;

PROC PRINT Statement

PROC PRINT *options*;

The options below can be specified in the PROC PRINT statement on all operating systems:

DATA=*SASdataset* names the SAS data set to be used by PROC PRINT. If DATA= is omitted, the most recently created data set is used.

N requests that the number of observations be printed at the end of the data set. If a BY statement is used, the number of observations in each BY group is printed at the end of the BY group.

UNIFORM
U requests that the values of a variable be printed in the same columns of each page in the output. Without the UNIFORM option, the PRINT procedure fits as many variables and observations on a page as possible. This can result in a different number of variables being printed on each page.

DOUBLE
D causes the printed output to be double-spaced.

ROUND rounds values to the number of decimal places specified for a variable in a FORMAT statement or, if no format is specified, to two decimal places. The only variables whose values are rounded are those being summed (see the SUM and SUMBY statements, below). Values are rounded before printing, and the rounded values are accumulated for printing of totals.

LABEL uses variables' labels as column headings. Labels can be specified in LABEL statements in the DATA step that creates the data set or in the PROC PRINT step. See "Statements Used in the PROC Step" for a description of the LABEL statement. If LABEL is not specified, or if there is no label for a variable, the variable name is used. If the LABEL option is specified and there is at least one variable with a label, column headings are not printed vertically; therefore, the number of output pages may be greater than if printed without the use of the LABEL option. PROC PRINT splits labels if necessary, in order to conserve space. Labels are printed for BY variables but are not printed in SUM lines (see the example output at the end of this chapter).

SPLIT='*splitchar*' splits labels used as column headings where the split character appears. When you define the value of the split character, you must enclose it in single quotes. It is not necessary to use both the LABEL and SPLIT=

options since SPLIT= implies that labels are to be used.
Below is an example. With these statements:

```
PROC PRINT DATA=CLASS SPLIT='*';
   LABEL X='THIS IS*A LABEL';
```

the label for X prints like this:

```
THIS IS
A LABEL
```

The split character is not printed. Blanks in the label are permitted. Up to three lines of labels are allowed; that is, no more than two split characters can be used. To print a column with no heading, use the split character as the variable's label. For example, these statements:

```
DATA TEST;
   INPUT X $ @@;
   LABEL X='*';
   CARDS;
   AAA BBB CCC DDD
   ;
PROC PRINT DATA=TEST SPLIT='*';
```

produce this output:

```
OBS
 1    AAA
 2    BBB
 3    CCC
 4    DDD
```

Labels of BY variables are not split, even if SPLIT= is specified. The split character is printed as part of the label.

NOOBS causes the observation number to be suppressed. The NOOBS option produces the same effect as the ID statement. It is most useful when you do not want to use an ID statement, but you do not want the observation numbers to be printed.

VAR Statement

VAR *variables;*

The VAR statement names the variables to be printed. The variables are printed in the order in which they appear in the VAR statement. If no VAR statement is used, all variables in the data set are printed.

ID Statement

ID *variables;*

The formatted values of the variables in the ID statement are used instead of observation numbers to identify observations in the output. More than one variable can be specified. When an observation is too long to print on one line, the values of the ID variables are printed at the beginning of every line containing

data values for the observation. The ID variables can be up to one-half the width of the page as determined by the current value of the LINESIZE= option,[1] which usually has a default value of 132.

BY Statement

BY *variables;*

A BY statement can be used with PROC PRINT to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

The LABEL and SPLIT= options have no effect on BY variables since the BY variable values are printed in a dashed line above the remaining printed output for the BY group.

PAGEBY Statement

PAGEBY *variable;*

The PAGEBY statement causes PROC PRINT to begin printing a new page when the specified BY variable changes value or when any BY variable listed before it in the BY statement changes value. For example,

```
PROC PRINT;
   BY X Y Z;
   PAGEBY Y;
```

causes SAS to print a new page when X or Y changes value, but not when Z changes value.

A BY statement is required with the PAGEBY statement.

SUM Statement

SUM *variables;*

The SUM statement specifies variables whose values are to be totaled. You can specify a variable in the SUM statement that is not listed on the VAR statement, and the procedure adds the variable to the VAR list. PRINT ignores requests for totals on BY and ID variables.

When a BY statement is used and only one BY variable is specified, the SUM variable is totaled for each BY group (subtotaled) containing more than one observation. Consider this example:

```
DATA A;
   INPUT X Y Z @@;
   CARDS;
1 1 1 1 1 2 2 1 1 2 1 2 3 1 1 3 3 1 3 3 2 3 4 1
;
PROC PRINT;
   BY X;
   SUM Z;
   TITLE 'ACCORDING TO VARIABLE X BY GROUPS';
```

Data set A is created and then printed with the Z variable values subtotaled according to variable X BY groups. Since the observations are already in sorted

order by the X and Y values, it is not necessary to use PROC SORT. Subtotals are printed for each BY group because they all contain more than one observation.

Output 43.1 Variable Values Subtotaled According to Variable X BY Groups

```
                          ACCORDING TO VARIABLE X BY GROUPS                    1

-------------------------------------------------------X=1-----------------------------------------------------

                                    OBS    Y    Z

                                     1     1    1
                                     2     1    2
                                     -          -
                                     X          3


-------------------------------------------------------X=2-----------------------------------------------------

                                    OBS    Y    Z

                                     3     1    1
                                     4     1    2
                                     -          -
                                     X          3


-------------------------------------------------------X=3-----------------------------------------------------

                                    OBS    Y     Z

                                     5     1     1
                                     6     3     1
                                     7     3     2
                                     8     4     1
                                     -          --
                                     X          5
                                                ==
                                                11
```

When a BY statement is used and multiple BY variables are specified, subtotals are printed for a BY variable only when it changes value and there is more than one observation with that value. (Any BY variables listed after a given BY variable change value when that BY variable changes value. This is shown in **Output 43.2**; when X changes value, Y changes value also.) For example, if the data set A (above) is printed with X and Y as BY variables:

```
PROC PRINT;
   BY X Y;
   SUM Z;
   TITLE 'ACCORDING TO VARIABLE X AND Y BY GROUPS';
```

The output produced by these statements is shown in **Output 43.2.**

Output 43.2 Variable Values Subtotaled According to Variable X and Y BY Groups

```
                          ACCORDING TO VARIABLE X AND Y BY GROUPS                          2

-------------------------------------------------------X=1 Y=1-------------------------------------------------------

                                                    OBS     Z

                                                     1      1
                                                     2      2
                                                     -      -
                                                     Y      3
                                                     X      3


-------------------------------------------------------X=2 Y=1-------------------------------------------------------

                                                    OBS     Z

                                                     3      1
                                                     4      2
                                                     -      -
                                                     Y      3
                                                     X      3


-------------------------------------------------------X=3 Y=1-------------------------------------------------------

                                                    OBS     Z

                                                     5      1


-------------------------------------------------------X=3 Y=3-------------------------------------------------------

                                                    OBS     Z

                                                     6      1
                                                     7      2
                                                     -      -
                                                     Y      3


-------------------------------------------------------X=3 Y=4-------------------------------------------------------

                                                    OBS     Z

                                                     8      1
                                                     -     --
                                                     X      5
                                                           ==
                                                           11
```

SUMBY Statement

SUMBY *byvariable;*

The SUMBY statement is used to print totals when the specified BY variable changes value or when any BY variable listed before it in the BY statement changes value. A BY statement is required with the SUMBY statement.

The variables that are totaled are those that appear in the SUM statement. If there is no SUM statement, the procedure totals all numeric variables in the data set except those listed in a BY statement.

These statements:

```
PROC PRINT;
   SUM A B C;
   BY X Y Z;
   SUMBY Y;
```

total the variables A, B, and C when either X or Y changes value, but not when Z changes value.

The effect of the SUMBY statement is to limit the printing of subtotals for BY groups; without it, subtotals are printed when any BY variable changes value and there is more than one observation with the value.

DETAILS

Usage Notes

Formats If you provide formats for the numeric variables (for example, with FORMAT statements), decimal points are not shifted each time a value is printed (that is, they align in the column). Numeric values not associated with a specific format are given a maximum width of 12 columns. (The FORMAT statement is described in "Statements Used in the PROC Step.")

Page format PROC PRINT formats one page at a time and, by default, attempts to use as few pages as possible. First, it attempts to print observations on a single line. Column headings may be rotated (printed vertically rather than horizontally) to accomplish this unless the SPLIT= or LABEL options are specified. If it cannot fit the observations on single lines, the observations are split into two or more sections on the page, and the observation number or ID variable(s) is printed at the beginning of each line.

The UNIFORM option The maximum number of observations that fit on a page is based on data widths, so the columns used to print certain variables can vary from page to page. The UNIFORM option prevents this, but it requires more pages to print a data set.

The data width of an unformatted character variable is its length or the pagesize minus the length of the ID variable, whichever is less. The width of an unformatted numeric variable is 12.

Printed Output

The printed output for PROC PRINT includes:

1. OBS, the number of each observation in the data set, appears in the leftmost column of the output. (OBS is not a variable in the data set; it is only used by the procedure to identify observations when the ID statement is not used or when NOOBS is not used.) If an ID variable is used, ID values rather than observation numbers are printed, and the column is headed with the name of the ID variable.
2. the names or labels of all variables that are printed.
3. the current value(s) of the BY variable, if any, above each section.
4. totals for the specified variables.

EXAMPLES

Printing Reports: Example 1

A company's sales and expense reports for four regions of the United States are printed in three different ways in this example. The data set is first created and a simple listing of the data is printed.

```
DATA BRANCH;
   INPUT REGION $ STATE $ MONTH MONYY5.
         HEADCNT EXPENSES REVENUE;
   FORMAT MONTH MONYY5.;
   CARDS;
EASTERN  VA FEB78  10  7800 15500
SOUTHERN FL MAR78   9  9800 13500
SOUTHERN GA JAN78   5  2000  8000
NORTHERN MA MAR78   3  1500  1000
SOUTHERN FL FEB78  10  8500 11000
NORTHERN NY MAR78   5  6000  5000
EASTERN  VA MAR78  11  8200 16600
PLAINS   NM MAR78   2  1350   500
SOUTHERN FL JAN78  10  8000 10000
NORTHERN NY FEB78   4  3000  4000
SOUTHERN GA FEB78   7  1200  6000
;
PROC PRINT;
   TITLE 'PROC PRINT PROVIDES A LISTING';
```

Output 43.3 A Listing of the Data Set

PROC PRINT PROVIDES A LISTING 3

OBS	REGION	STATE	MONTH	HEADCNT	EXPENSES	REVENUE
1	EASTERN	VA	FEB78	10	7800	15500
2	SOUTHERN	FL	MAR78	9	9800	13500
3	SOUTHERN	GA	JAN78	5	2000	8000
4	NORTHERN	MA	MAR78	3	1500	1000
5	SOUTHERN	FL	FEB78	10	8500	11000
6	NORTHERN	NY	MAR78	5	6000	5000
7	EASTERN	VA	MAR78	11	8200	16600
8	PLAINS	NM	MAR78	2	1350	500
9	SOUTHERN	FL	JAN78	10	8000	10000
10	NORTHERN	NY	FEB78	4	3000	4000
11	SOUTHERN	GA	FEB78	7	1200	6000

Now sort the data set and use PRINT to create a report showing revenue and expense totals.

```
PROC SORT;
   BY REGION STATE MONTH;
PROC PRINT SPLIT='*';
   LABEL REGION='SALES REGION'
         HEADCNT='SALES*PERSONNEL';
   BY REGION STATE;
   SUM REVENUE EXPENSES;
   TITLE 'REVENUE AND EXPENSE TOTALS';
```

Output 43.4 Sorted to Show Revenue and Expense Totals

```
                                  REVENUE AND EXPENSE TOTALS                                    4
                                  ❸
-----------------------------------------SALES REGION =EASTERN STATE=VA-----------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

   1      FEB78        10           7800        15500
   2      MAR78        11           8200        16600
 ------                                ❹       -----       -----
 STATE                                         16000       32100
 REGION                                        16000       32100


-----------------------------------------SALES REGION =NORTHERN STATE=MA-----------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

   3      MAR78        3            1500        1000


-----------------------------------------SALES REGION =NORTHERN STATE=NY-----------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

   4      FEB78        4            3000        4000
   5      MAR78        5            6000        5000
 ------                            -----       -----
 STATE                              9000        9000
 REGION                            10500       10000


-----------------------------------------SALES REGION =PLAINS STATE=NM-------------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

   6      MAR78        2            1350        500


-----------------------------------------SALES REGION =SOUTHERN STATE=FL-----------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

   7      JAN78        10           8000        10000
   8      FEB78        10           8500        11000
   9      MAR78        9            9800        13500
 ------                            -----       -----
 STATE                             26300       34500
```

```
                                  REVENUE AND EXPENSE TOTALS                                    5

-----------------------------------------SALES REGION =SOUTHERN STATE=GA-----------------------------------------

                                  SALES
 OBS      MONTH     PERSONNEL     EXPENSES     REVENUE

  10      JAN78        5            2000        8000
  11      FEB78        7            1200        6000
 ------                            -----       -----
 STATE                              3200        14000
 REGION                            29500        48500
                                   =====       =====
                                   57350       91100
```

A different type of summary report is printed when the same variables are specified in both ID and BY statements.

```
PROC PRINT SPLIT='*';
   LABEL REGION='SALES REGION'
         HEADCNT='SALES*PERSONNEL';
   BY REGION STATE;
   ID REGION STATE;  /* PUT EACH BY VARIABLE IN ID */
   SUM REVENUE EXPENSES;
   VAR REVENUE EXPENSES HEADCNT MONTH;
   FORMAT HEADCNT 3. REVENUE EXPENSES COMMA10.;
   TITLE 'BRANCH HEADCOUNT, SALES, AND EXPENSES';
```

Output 43.5 Branch Headcount, Sales, and Expenses

BRANCH HEADCOUNT, SALES AND EXPENSES 6

SALES REGION	STATE	REVENUE	EXPENSES	SALES PERSONNEL	MONTH
EASTERN	VA	15,500	7,800	10	FEB78
		16,600	8,200	11	MAR78
EASTERN	VA	32,100	16,000		
EASTERN		32,100	16,000		
NORTHERN	MA	1,000	1,500	3	MAR78
NORTHERN	NY	4,000	3,000	4	FEB78
		5,000	6,000	5	MAR78
NORTHERN	NY	9,000	9,000		
NORTHERN		10,000	10,500		
PLAINS	NM	500	1,350	2	MAR78
SOUTHERN	FL	10,000	8,000	10	JAN78
		11,000	8,500	10	FEB78
		13,500	9,800	9	MAR78
SOUTHERN	FL	34,500	26,300		
SOUTHERN	GA	8,000	2,000	5	JAN78
		6,000	1,200	7	FEB78
SOUTHERN	GA	14,000	3,200		
SOUTHERN		48,500	29,500		
		91,100	57,350		

Vertical Column Headings: Example 2

This example shows how rotation of column headings occurs to help keep the output on a smaller number of pages.

```
DATA SMALL;
   DO I=1 TO 5;
      ARRAY  X{*} X1-X50;
      DO J=1 TO 50;
         X{J}=MOD(J,5);
         END;
      OUTPUT;
      END;
PROC PRINT;
   TITLE 'PRINT WILL ROTATE TITLES IF IT HELPS';
```

Output 43.6 Vertical Column Headings

```
                                    PRINT WILL ROTATE TITLES IF IT HELPS                                     7

O                     X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X
B   X X X X X X X X X 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5
S I 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 J

1 1 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 51
2 2 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 51
3 3 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 51
4 4 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 51
5 5 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 51
```

NOTE

1. **AOS/VS, CMS, PRIMOS, VM/PC, and VMS**: Use the TLINESIZE= option when executing SAS under these operating systems.

Chapter 44

The PRINTTO Procedure

Operating systems: CMS, OS, VM/PC, and VSE

ABSTRACT

Use the PRINTTO procedure to define a destination for SAS procedure output.

INTRODUCTION

Normally, SAS procedure output is routed to the *standard SAS print file*, which is the file referenced by the fileref FT*nn*F001, where *nn* is the second number of the SAS system option UNITS=. Under batch systems, the usual destination for the standard print file is a printer. Under interactive systems, your terminal is the usual destination.[1]

The standard print file is usually defined automatically for you by your installation. Refer to the chapter, "SAS Log and Procedure Output" for a complete discussion of output from SAS procedures.

You can use PROC PRINTTO to:

- choose your output destination
- write SAS output on tape for COM (computer output to microfiche)
- selectively suppress SAS output by routing output to a DUMMY file
- print several copies of SAS output
- route the print file to a special printer or forms
- route the print file to a permanent file
- use SAS output as input data within the same job (if you also write some program statements).

SPECIFICATIONS

The PROC PRINTTO statement is the only statement necessary.

PROC PRINTTO *options*;

PROC PRINTTO Statement

PROC PRINTTO *options*;

The options below can appear in the PROC PRINTTO statement.

UNIT=*nn* defines the unit number of the FORTRAN-style fileref that references the output file in the control language for your job or session. For example, UNIT=20 specifies that the output is written to the file with the fileref FT20F001. The UNIT= option must be used when procedure output is suppressed or routed to a file other than the standard SAS print file.

Do not use any unit number that has been specified in the global SAS system option UNITS=, unless you are certain that it will not be used for anything else. Ordinarily the SAS System uses the first five unit numbers defined by UNITS=, which are typically 11, 12, 13, 14, and 15.

When you have changed the destination for the SAS print file, use a PROC PRINTTO statement without the UNIT= option to route the output back to the standard print file.

NEW indicates that you want to write over any existing output.[2] If you do not use the NEW option, the output is appended to any existing output in the file. For example, suppose output is currently routed to the printer. To route the output to FT18F001, use the statement:

```
PROC PRINTTO UNIT=18 NEW;
```

The statement

```
PROC PRINTTO;
```

routes the output back to the printer. To send more output to FT18F001, use the statement:

```
PROC PRINTTO UNIT=18;
```

Output from this point on is appended to the earlier output.

If you want subsequent output to be written **over** (replace) earlier output, repeat the NEW specification with the PROC PRINTTO statement when you redefine the unit as the output destination.

DETAILS

Output

PRINTTO does not produce an output data set or printed output.

Usage Notes

- If you are not familiar with the control language needed to define output files, consult the appropriate documentation for your operating system.
- PRINTTO is useful when you want to route output from different steps of a SAS job to different files. When the global SAS system option, NUMBER, is in effect, there is one page numbering sequence for all output from the job. That is, the page number is **not** set back to 1 when output is routed to a new file.
- Default values for the output file DCB attributes are used unless they are defined in the control language.[3]

EXAMPLES

Writing FREQ Output to Tape: Example 1

The first PRINTTO statement changes the standard SAS print file (in this case, the printer) to another file, to which PROC FREQ output is routed. The second PRINTTO statement routes PROC CORR output back to the standard SAS print file. Notice that the page number on the PROC CORR output is 121, rather than 1. This is because there is only one sequence of page numbers in the job.

```
DATA NUMBERS;
   INPUT X Y Z;
   CARDS;
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
 4.2 16.8 129.5
 6.0 24.9 157.0
10.2 39.6 187.9
11.7 31.1 140.5
 7.2 25.5 128.0
 5.5 19.4  39.6
 9.9 21.8 211.3
 7.4 26.5 123.2
 2.3 10.6  41.2
 6.6 22.0 100.7
10.1 19.1  81.1
15.5 30.9 142.9
 2.4 13.5  38.7
 8.0 34.8 292.1
 ;
PROC PRINTTO NEW UNIT=20;
   TITLE 'PROC FREQ OUTPUT DIRECTED TO UNIT 20';
PROC FREQ;
   TABLES X*Y*Z;
PROC PRINTTO;
PROC CORR;
   TITLE 'PROC CORR OUTPUT RE-DIRECTED TO STANDARD PRINT FILE';
```

Output 44.1 Output to Unit 20 and the Standard Print File

```
                    PROC CORR OUTPUT RE-DIRECTED TO STANDARD PRINT FILE                    1

VARIABLE      N          MEAN          STD DEV             SUM          MINIMUM          MAXIMUM

X            20    8.40500000    3.51934130     168.10000000     2.30000000    15.50000000
Y            20   28.32500000   11.09347248     566.50000000    10.60000000    51.60000000
Z            20  134.32000000   70.83407298    2686.40000000    38.70000000   292.10000000


          CORRELATION COEFFICIENTS / PROB > |R| UNDER H0:RHO=0 /  N = 20

                                    X         Y         Z

                    X         1.00000   0.53435   0.35346
                              0.0000    0.0152    0.1263

                    Y         0.53435   1.00000   0.59041
                              0.0152    0.0000    0.0061

                    Z         0.35346   0.59041   1.00000
                              0.1263    0.0061    0.0000
```

Printing Several Copies of Procedure Output: Example 2

This example prints multiple copies of output from PROC FREQ and PROC MEANS. The output is written to a disk file referenced by FT22F001 and printed the number of times specified in the PR macro call.

```
DATA RATES;
   INPUT RATE1-RATE3;
   CARDS;
16 27 27
18 20 25
15 15 31
15 32 32
12 15 16
20 23 23
24 24 25
21 25 23
27 45 24
12 13 15
22 32 33
31 32 33
29 24 26
34 32 28
26 25 23
53 48 75
;
PROC PRINTTO UNIT=22 NEW;
PROC FREQ;
   TITLE 'MULTIPLE COPIES OF PROC FREQ AND PROC MEANS OUTPUT';
PROC MEANS;
PROC PRINTTO;
%MACRO PR(COPIES);
   %DO I=1 %TO &COPIES;
```

```
    DATA _NULL_;
    FILE PRINT NOPRINT;
    INFILE FT22F001;
    INPUT;
    PUT _INFILE_;
    %END;
  %MEND PR;
%PR(2);
```

Output 44.2 Multiple Copies of PROC FREQ and PROC MEANS Output

MULTILPLE COPIES OF PROC FREQ AND MEANS OUTPUT 1

RATE1	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
12	2	2	12.500	12.500
15	2	4	12.500	25.000
16	1	5	6.250	31.250
18	1	6	6.250	37.500
20	1	7	6.250	43.750
21	1	8	6.250	50.000
22	1	9	6.250	56.250
24	1	10	6.250	62.500
26	1	11	6.250	68.750
27	1	12	6.250	75.000
29	1	13	6.250	81.250
31	1	14	6.250	87.500
34	1	15	6.250	93.750
53	1	16	6.250	100.000

RATE2	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
13	1	1	6.250	6.250
15	2	3	12.500	18.750
20	1	4	6.250	25.000
23	1	5	6.250	31.250
24	2	7	12.500	43.750
25	2	9	12.500	56.250
27	1	10	6.250	62.500
32	4	14	25.000	87.500
45	1	15	6.250	93.750
48	1	16	6.250	100.000

RATE3	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
15	1	1	6.250	6.250
16	1	2	6.250	12.500
23	3	5	18.750	31.250
24	1	6	6.250	37.500
25	2	8	12.500	50.000
26	1	9	6.250	56.250
27	1	10	6.250	62.500
28	1	11	6.250	68.750
31	1	12	6.250	75.000
32	1	13	6.250	81.250
33	2	15	12.500	93.750
75	1	16	6.250	100.000

MULTILPLE COPIES OF PROC FREQ AND MEANS OUTPUT 2

VARIABLE	N	MEAN	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	STD ERROR OF MEAN	SUM	VARIANCE	C.V.
RATE1	16	23.43750000	10.33420695	12.00000000	53.00000000	2.58355174	375.0000000	106.7958333	44.093
RATE2	16	27.00000000	9.79795897	13.00000000	48.00000000	2.44948974	432.0000000	96.0000000	36.289
RATE3	16	28.68750000	13.42494569	15.00000000	75.00000000	3.35623642	459.0000000	180.2291667	46.797

```
                    MULTILPLE COPIES OF PROC FREQ AND MEANS OUTPUT                    1

 RATE1   FREQUENCY  CUM FREQ   PERCENT  CUM PERCENT

    12         2         2    12.500       12.500
    15         2         4    12.500       25.000
    16         1         5     6.250       31.250
    18         1         6     6.250       37.500
    20         1         7     6.250       43.750
    21         1         8     6.250       50.000
    22         1         9     6.250       56.250
    24         1        10     6.250       62.500
    26         1        11     6.250       68.750
    27         1        12     6.250       75.000
    29         1        13     6.250       81.250
    31         1        14     6.250       87.500
    34         1        15     6.250       93.750
    53         1        16     6.250      100.000

 RATE2   FREQUENCY  CUM FREQ   PERCENT  CUM PERCENT

    13         1         1     6.250        6.250
    15         2         3    12.500       18.750
    20         1         4     6.250       25.000
    23         1         5     6.250       31.250
    24         2         7    12.500       43.750
    25         2         9    12.500       56.250
    27         1        10     6.250       62.500
    32         4        14    25.000       87.500
    45         1        15     6.250       93.750
    48         1        16     6.250      100.000

 RATE3   FREQUENCY  CUM FREQ   PERCENT  CUM PERCENT

    15         1         1     6.250        6.250
    16         1         2     6.250       12.500
    23         3         5    18.750       31.250
    24         1         6     6.250       37.500
    25         2         8    12.500       50.000
    26         1         9     6.250       56.250
    27         1        10     6.250       62.500
    28         1        11     6.250       68.750
    31         1        12     6.250       75.000
    32         1        13     6.250       81.250
    33         2        15    12.500       93.750
    75         1        16     6.250      100.000
```

```
                    MULTILPLE COPIES OF PROC FREQ AND MEANS OUTPUT                    2

VARIABLE      N        MEAN       STANDARD      MINIMUM       MAXIMUM     STD ERROR        SUM        VARIANCE       C.V.
                                 DEVIATION       VALUE         VALUE      OF MEAN

RATE1        16  23.43750000  10.33420695  12.00000000  53.00000000  2.58355174  375.0000000  106.7958333      44.093
RATE2        16  27.00000000   9.79795897  13.00000000  48.00000000  2.44948974  432.0000000   96.0000000      36.289
RATE3        16  28.68750000  13.42494569  15.00000000  75.00000000  3.35623642  459.0000000  180.2291667      46.797
```

Putting Statistics from GLM into a SAS Data Set: Example 3

In this example, GLM output is written to a temporary disk file with the fileref FT20F001. Next a DATA step:

- reads the FT20F001 file.
- writes the FT20F001 file. Notice that the output from FILE PRINT looks exactly as the PROC GLM output would look if PROC PRINTTO had not been used.
- creates a SAS data set called EMS. EMS contains the error mean square values from the GLM output statistics.

```
DATA EXAMPLE;
   INPUT Y1-Y3 X1-X3;
   CARDS;
 57.2   96.0  83.0  50.0   22.0  81.9
 22.5    0.0   0.0  55.0    0.0  63.6
```

```
 15.2   62.0  16.0  62.0   67.0  29.8
 17.8   17.0   0.0   0.0    0.0  55.3
  3.6  100.0  17.0  17.0   28.0  78.0
188.0   17.0   0.0   0.0    0.0  62.0
 46.9   17.0   0.0 100.0    0.0  12.0
 63.8   33.0   0.0  84.0    0.0  18.3
  2.5  100.0  17.0  82.8   67.0   4.6
250.0    0.0   0.0  12.0    0.0   8.8
250.0   17.0   0.0   0.0    0.0  21.3
 67.4   78.0   0.0  33.0   12.0  16.7
 49.3   23.0  48.0   0.0   54.0  33.5
155.0    0.0  12.3  83.0    0.0  22.8
250.0    2.0   0.0   0.0    0.0   9.5
250.0   65.0  99.8  32.6    0.0  19.1
192.0    0.7  16.0  77.0   22.0  12.8
250.0    4.0   0.0   0.0   11.0  19.3
250.0   33.0  57.4  23.5   11.0  18.0
250.0   61.0  17.0   0.7    0.0   1.9
250.0   17.0   9.0  59.0    6.0   4.9
250.0   44.0   0.0  29.5   11.0   6.0
250.0    0.0  24.0   0.4    0.0  13.0
123.0   19.0   0.0  33.0   33.0   5.0
250.0   33.0  17.0   0.8    0.0   2.0
350.0   31.0  17.0 100.0   54.0   9.8
250.0   17.0  90.0   4.9    0.0  17.0
540.0   99.0  78.0  46.2    0.0  21.0
250.0   17.0  55.0   5.2   67.3  54.0
250.0    0.0  89.0   0.0    0.0  12.0
;
PROC PRINTTO UNIT=20 NEW;
PROC GLM;
   MODEL Y1-Y3=X1-X3;
   TITLE 'PROC GLM OUTPUT DIRECTED TO UNIT 20';
PROC PRINTTO;
DATA EMS;
   FILE PRINT NOPRINT;
   INFILE FT20F001;
   INPUT @2 NAME $ @;
   PUT _INFILE_;
   IF NAME='ERROR' THEN DO;
      INPUT @2 NAME $12. DUM1 DUM2 ERRORMS;
      KEEP ERRORMS;
      OUTPUT;
      END;
PROC PRINT;
   TITLE 'EMS DATA SET PRINTED TO THE STANDARD PRINT FILE';
PROC MEANS;
   TITLE 'PROC MEANS OUTPUT DIRECTED TO THE STANDARD PRINT FILE';
```

Output 44.3 Output to Unit 20 and the Standard Print File

PROC GLM OUTPUT DIRECTED TO UNIT 20 1
GENERAL LINEAR MODELS PROCEDURE

DEPENDENT VARIABLE: Y1

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F	R-SQUARE	C.V.
MODEL	3	126089.40702688	42029.80234229	3.39	0.0328	0.281363	61.8967
ERROR	26	322047.95163978	12386.45967845		ROOT MSE		Y1 MEAN
CORRECTED TOTAL	29	448137.35866667			111.29447281		179.80666667

SOURCE	DF	TYPE I SS	F VALUE	PR > F	DF	TYPE III SS	F VALUE	PR > F
X1	1	21242.08290601	1.71	0.2018	1	21302.15024071	1.72	0.2012
X2	1	25738.72420078	2.08	0.1614	1	12090.33501164	0.98	0.3323
X3	1	79108.59992010	6.39	0.0179	1	79108.59992010	6.39	0.0179

PARAMETER	ESTIMATE	T FOR H0: PARAMETER=0	PR > |T|	STD ERROR OF ESTIMATE
INTERCEPT	278.83736593	7.32	0.0001	38.11127089
X1	-0.82798292	-1.31	0.2012	0.63136900
X2	-0.92577857	-0.99	0.3323	0.93704738
X3	-2.34245584	-2.53	0.0179	0.92690059

PROC GLM OUTPUT DIRECTED TO UNIT 20 2
GENERAL LINEAR MODELS PROCEDURE

DEPENDENT VARIABLE: Y2

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F	R-SQUARE	C.V.
MODEL	3	4786.31777135	1595.43925712	1.50	0.2384	0.147385	97.6369
ERROR	26	27688.59589532	1064.94599597		ROOT MSE		Y2 MEAN
CORRECTED TOTAL	29	32474.91366667			32.63351032		33.42333333

SOURCE	DF	TYPE I SS	F VALUE	PR > F	DF	TYPE III SS	F VALUE	PR > F
X1	1	1171.18658150	1.10	0.3040	1	719.49088009	0.68	0.4186
X2	1	2296.77406784	2.16	0.1539	1	1681.60067002	1.58	0.2201
X3	1	1318.35712201	1.24	0.2760	1	1318.35712201	1.24	0.2760

PARAMETER	ESTIMATE	T FOR H0: PARAMETER=0	PR > |T|	STD ERROR OF ESTIMATE
INTERCEPT	15.64104937	1.40	0.1734	11.17489953
X1	0.15216764	0.82	0.4186	0.18512857
X2	0.34526248	1.26	0.2201	0.27475889
X3	0.30239604	1.11	0.2760	0.27178367

PROC GLM OUTPUT DIRECTED TO UNIT 20 3
GENERAL LINEAR MODELS PROCEDURE

DEPENDENT VARIABLE: Y3

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F	R-SQUARE	C.V.
MODEL	3	1239.83264718	413.27754906	0.36	0.7813	0.040043	133.0275
ERROR	26	29723.04901948	1143.19419306		ROOT MSE		Y3 MEAN
CORRECTED TOTAL	29	30962.88166667			33.81115486		25.41666667

SOURCE	DF	TYPE I SS	F VALUE	PR > F	DF	TYPE III SS	F VALUE	PR > F
X1	1	707.99336799	0.62	0.4384	1	680.41593391	0.60	0.4474
X2	1	214.66351794	0.19	0.6683	1	129.70942315	0.11	0.7389
X3	1	317.17576125	0.28	0.6028	1	317.17576125	0.28	0.6028

PARAMETER	ESTIMATE	T FOR H0: PARAMETER=0	PR > |T|	STD ERROR OF ESTIMATE
INTERCEPT	25.19209083	2.18	0.0388	11.57816781
X1	-0.14797791	-0.77	0.4474	0.19180930
X2	0.09589009	0.34	0.7389	0.28467410
X3	0.14832336	0.53	0.6028	0.28159152

EMS DATA SET PRINTED TO THE STANDARD PRINT FILE 4

OBS	ERRORMS
1	12386.5
2	1064.9
3	1143.2

PROC MEANS OUTPUT DIRECTED TO THE STANDARD PRINT FILE 5

VARIABLE	N	MEAN	STANDARD DEVIATION	MINIMUM VALUE	MAXIMUM VALUE	STD ERROR OF MEAN	SUM	VARIANCE	C.V.
ERRORMS	3	4864.866622	6514.008157	1064.945996	12386.45968	3760.864363	14594.59987	42432302.27	133.899

NOTES

1. **CMS**: the standard print file is usually routed to a disk file under noninteractive CMS.

2. **VSE**: the NEW option is assumed for a disk or tape file. Output cannot be appended to a disk or tape file.

3. **OS**: the default values are RECFM=VBA, LRECL=137 and BLKSIZE=6391. The RECFM attribute must include an A because the print file contains carriage control characters.

   CMS: the default BLKSIZE= value is 141.

   VSE: the values are the same as for the standard SAS print file: RECFM=VBA, LRECL=value of the global SAS system option LINESIZE, BLKSIZE=FILEBLKSIZE (devicetype). You cannot specify different values in the control language.

Chapter 45

The RELEASE Procedure

Operating system: OS

ABSTRACT

In an OS operating environment the RELEASE procedure releases unused space at the end of a disk data set.

INTRODUCTION

The RELEASE procedure releases space that is allocated to a disk data set, but not used. Only unused space at the end of a data set can be released. If you delete members from a SAS library, for example, the library may contain embedded unused space. You cannot use RELEASE to release embedded space. In other words, you can release space only after the HIGH TRACK USED, as indicated by the CONTENTS or DATASETS procedures.

In the control language you can release unused space with specifications such as SPACE=(,,RLSE) in the DD statement under OS batch, or you can use the RELEASE operand of the TSO ALLOCATE command. However, releasing unused space with PROC RELEASE offers several advantages over methods provided by the operating system. For instance, with PROC RELEASE, the user, not the operating system, controls when unused space is released. This advantage is especially applicable to TSO users. If you specify RELEASE on the ALLOCATE command under TSO, unused space is released each time the data set is closed. If you execute commands or programs that cause the data set to be closed many times during a single TSO session, the limit of sixteen extents can be quickly exceeded. To avoid this problem use PROC RELEASE.

Another advantage of PROC RELEASE is that you can control the total number of tracks finally allocated to a data set with PROC RELEASE options. There is no danger of erasing all or part of a data set because RELEASE frees unused space only. PROC RELEASE returns unused space to the pool of available space on the disk volume. Once released, the space is still available for allocation to the data set, provided that a secondary space allocation is given for the data set in the

control language and provided that all free space on the volume is not subsequently allocated to another data set.

PROC RELEASE can be used with any sequential, partitioned, or direct access data set, not just one containing SAS data sets. PROC RELEASE cannot be used to release unused space from the SAS WORK library or to release space from ISAM or VSAM data sets.

With options on the PROC RELEASE statement, you can specify the exact number of tracks to be allocated to a data set or the amount of unused space you want to keep or release.

SPECIFICATIONS

The PROC statement is the only statement used.

PROC RELEASE DDNAME=*fileref options***;**

PROC RELEASE Statement

PROC RELEASE DDNAME=*fileref* [*options*];

DDNAME=*fileref*
: refers to the data set from which unused space is to be released. DDNAME= must be specified.

options
: specifies the amount of unused space to be released from the data set. Use only one of the following four options:

TOTAL=*number*
TRACKS=*number*
: specifies the total *number* of tracks that the data set should contain after unused space is released, that is, after PROC RELEASE has executed. For example, the statement:

```
PROC RELEASE DDNAME=SURVEY TOTAL=10;
```

releases all but ten tracks for the data set referenced by the fileref SURVEY.

The procedure calculates the amount of space to be released in the following way: amount of space allocated − (TOTAL=value) = amount of unused space released.

If the value you specify is smaller than the amount of used space in the data set, the SAS System releases only the unused space at the end of the data set.

UNUSED=
: specifies the number of tracks of unused space that the data set should contain after PROC RELEASE has executed. The procedure calculates the amount of unused space to be released in the following way: Amount of space allocated − (used space + UNUSED=value) = amount of unused space released.

If the value you specify is greater than the amount of unused space in the data set, no space is released at the end of the data set.

RELEASE=
: specifies the number of tracks of unused space that is to be released. If the value you specify is greater than the amount of unused space in the data set, the SAS System releases all of the unused space at the end of the data set.

EXTENTS
EXTENT
EX

requests that only the space allocated to completely unused secondary extents be released. After the procedure releases unused space from the data set, the size of the data set is the sum of the primary extent plus all used secondary extents.

If you do not specify one of the above options in the PROC RELEASE statement, all unused space at the end of the data set is released.

With the following option you specify the unit boundary on which the data set should end.

BOUNDARY=*type*
TYPE=*type*

specifies whether the data set will end on a track or cylinder boundary.

After the total amount of space to be retained is calculated, this amount is rounded up to the next unit boundary. Any remaining space is then released. Remember that the total amount of space will include that which is actually used and may also include unused space requested with other options. BOUNDARY=*type* then will increase the amount of unused space retained in the data set by the portion of the unit required to reach (or round up to) the next boundary. *Type* can be:

DSCB | DATASET

specifies that the data set will end on the next track or cylinder boundary depending on how space is currently allocated. If allocated in tracks, the total amount of space to be retained is calculated, and remaining unused tracks are released. If allocated in cylinders, the space to be retained is rounded up to the next cylinder boundary, then remaining unused space is released. BOUNDARY=DSCB is the default.

CYL | CYLS | CYLINDER | CYLINDERS

specifies that space to be retained is rounded to the next cylinder boundary before remaining unused space is released. This specification is effective only if the data set is currently allocated in multiples of cylinders.

TRK | TRKS | TRACK | TRACKS

specifies that unused tracks are to be released. Since the minimum unit of space that can be released is a track, the space to be retained is not rounded up.

JCL | DD | ALLOC

specifies that space to be retained is rounded to the next unit boundary (tracks or cylinders) depending upon the allocation unit specified in the job control language or TSO ALLOCATE command. For example

```
//DD2 DD DISP=OLD,DSN=MY.DATA,SPACE=(CYL,2)
```

in combination with BOUNDARY=DD is equivalent to specifying BOUNDARY=CYL.

DETAILS

Usage Notes

RELEASE issues an error message if you attempt to release space in the current SAS WORK data set, a SAS data library that is currently open or being used, either of the SAS macro libraries, the temporary LIBRARY data set that stores PROC FORMAT output for the duration of the job step, or any other data set that the SAS System determines to be currently open or in use.

When PROC RELEASE is invoked, the operating system's disk space management function (DADSM) must be able to obtain exclusive control of the data set. If it cannot, no indication that DADSM does not have control is passed to the SAS System, no space is released from the data set, and no error message is issued by the SAS System. If the messages on the SAS log indicate that no space was released from the data set, you should check to see if the data set is allocated to another job or to another user.

Printed Output

PROC RELEASE prints a message on the SAS log giving the number of tracks allocated to the data set before and after execution of RELEASE, the number of tracks used, and the number of extents used.

EXAMPLE

The example below releases the unused secondary extents for an OS data set referenced by the fileref LIB.

```
// EXEC SAS
//LIB DD DISP=SHR,DSN=XXXXXX.YYYYYYY.ZZZZ
PROC RELEASE DDNAME=LIB EXTENTS;
```

REFERENCES

International Business Machines Corporation, *MVS JCL Reference*, Order Number GC28-1300.

International Business Machines Corporation, *MVS/XA JCL Reference*, Order Number GC28-1148.

International Business Machines Corporation, *OS/VS1 JCL Services*, Order Number GC24-5100.

Chapter 46

The SORT Procedure

Operating systems: All

ABSTRACT

The SORT procedure sorts observations in a SAS data set by one or more variables, storing the resulting sorted observations in a new SAS data set or replacing the original.

INTRODUCTION

PROC SORT is used most often for sorting a data set so that other SAS procedures can process it in subsets using BY statements. Data sets must also be sorted before they can be match-merged or updated.

The way PROC SORT operates depends on the operating system and the sort utility being used.[1] For many PROC SORT applications, the sort utility used makes no difference. However, if you need to know more about the sort utility or utilities at your installation, or where to find documentation on sort utilites, check with your installation's technical staff.

PROC SORT rearranges the observations in the data set according to the values of the variables in the BY statement, which **must** accompany the PROC SORT statement (see below). Suppose you want to sort a data set by an ID value that occurs in every observation. PROC SORT rearranges the data set so that the observation with the lowest ID value is first, the observation with the second lowest ID value is second, and so on. PROC SORT can also arrange observations in descending order.

When you want to sort by two or more variables, PROC SORT first arranges the data set in the order of the first BY variable. Then SORT arranges the observations having the lowest value of the first variable (if several observations have the same value) in the order of the second variable. This continues for every BY variable specified. For example, if you sort a data set containing state and city

information by state and then by city, SORT first arranges the data so that Alabama's observations are first, then Alaska's, and so on. Next, SORT arranges Alabama's observations by the city value, so that Birmingham's observations are followed by Dothan's, and so on.[2]

For numeric variables, the smallest-to-largest comparison sequence is

special missing value ._
SAS system missing value .
special missing value .A to .Z
negative numeric values
zero
positive numeric values

See "SAS System Options" for complete descriptions of SAS system options mentioned in this chapter.

SPECIFICATIONS

The statements used with PROC SORT are

PROC SORT *options*;
BY *option variable option variable ...*;

PROC SORT Statement

PROC SORT *options*;

The options below can appear in the PROC SORT statement. If the SAS data set that you want to sort is on tape, you **must** include the OUT= option.

Not all sort utilities (for example, SORTPGM=SAS) support all of the PROC SORT statement options. Consult your installation representative about the options that are pertinent to your system's sort utility and about valid values to specify for those options.

DATA=*SASdataset*
: names the SAS data set that you want to sort. If DATA= is omitted, the most recently created SAS data set is used.

  Operating systems: All

OUT=*SASdataset*
: specifies a name for the output data set. If OUT= is omitted, the DATA= data set is sorted and the sorted version replaces the original data set. If you want the OUT= data set to be permanent, specify a two-level name (see "SAS Files").

  Operating systems: All

EQUALS
NOEQUALS
: specifies the order of the observations in the output data set. EQUALS specifies that observations with identical BY variable values are to retain the same relative positions in the output data set as in the input data set. NOEQUALS specifies that this restriction is not necessary.

  Operating systems: All

NODUPLICATES
NODUP
checks for and eliminates duplicate records. Duplicate records frequently occur when the data you are sorting were derived from log or journal files that were accidentally dumped more than once.

This option causes PROC SORT to compare all variable values for each observation to the previous one written to the output data set. If an exact match is found, the record is not written to the output data set. A message is produced on the SAS log, indicating the number of duplicate records dropped.

Operating systems: All

The next four options are used to specify a different collating sequence:

NATIONAL
specifies that character variables are to be sorted using an alternate collating sequence, as defined by your installation, to reflect a country's National Symbol Use Differences (see the examples under the DANISH option). Your installation must have customized PROC SORT at the time SAS was installed to reflect national use differences.

Operating systems: CMS, OS, and VSE

REVERSE
specifies that character variables are to be sorted using a collating sequence that is reversed from the normal EBCDIC collating sequence.

This option is similar to the BY statement DESCENDING option; the difference is that DESCENDING can be used with both character and numeric variables.

Operating systems: CMS, OS, and VSE

DANISH
NORWEGIAN
specifies that character variables are to be sorted using the alternate collating sequence matching the Danish and Norwegian National Use Differences standard.

In the Danish and Norwegian National Use Differences alternate collating sequence, the standard graphics shown in **Table 46.1** are redefined.

Thus, the Danish/Norwegian collating sequence is (in part):

a,b,c,...,x,y,z,æ,ø,å,A,B,C,...,X,Y,Z,Æ,Ø,Å

Operating systems: CMS, OS, and VSE

Table 46.1 Danish Norwegian National Use Differences

UPPERCASE				LOWERCASE			
HEX	Standard Graphic	Alternate Collating Sequence	National Use Graphic	HEX	Standard Graphic	Alternate Collating Sequence	National Use Graphic
7B	#	EA	Æ	C0	{	AA	æ
7C	@	EB	Ø	6A	¦	AB	ø
5B	$	EC	Å	D0	}	AC	å

FINNISH
SWEDISH

specifies that character variables be sorted using an alternate collating sequence matching the Finnish and Swedish National Use Differences standard.

In the Finnish and Swedish National Use Differences alternate collating sequence, the standard graphics shown in **Table 46.2** are redefined.

Table 46.2 Finnish and Swedish National Use Differences

UPPERCASE				LOWERCASE			
HEX	Standard Graphic	Alternate Collating Sequence	National Use Graphic	HEX	Standard Graphic	Alternate Collating Sequence	National Use Graphic
5B	$	EA	Å	D0	}	AA	å
7B	#	EB	Ä	C0	{	AB	ä
7C	@	EC	Ö	6A	¦	AC	ö

Thus, the Finnish/Swedish collating sequence is (in part):

a,b,c,...,x,y,z,å,ä,ö,A,B,C,...,X,Y,Z,Å,Ä,Ö

Operating systems: CMS, OS, and VSE

The following options are rarely needed:

MESSAGE
M

prints a summary of the system sort utility's actions. This option is the default action if the SAS system option SORTMSG is in effect. MESSAGE is useful if you run PROC SORT and the SAS log prints a message that the sort did not work properly. Explanations of the messages can be found in the IBM or vendor reference manual that describes your system sort utility.

Operating systems: CMS, OS, and VSE

LIST
L
: provides additional information about the system sort. Not all sort utilities support the specification of the LIST option; they may require that it be specified at the time the sort utility is generated or installed. This option is the default action if the SAS system option SORTLIST is in effect.[3]

  Operating systems: CMS, OS, and VSE

LEAVE=*n*
: specifies the number of bytes to leave unallocated in the region. Occasionally, the SORT procedure runs out of main storage. If this happens, rerun the job increasing the LEAVE= value by 30000.[4]

  Operating systems: CMS and OS

TECHNIQUE=*xxxx*
T=*xxxx*
: specifies a four-character sort technique to be passed to the system sort utility. SAS does not check the validity of the specified value in any way, so you must be sure that it is correct.

  Operating systems: CMS and OS

SORTWKNO=*number*
: specifies the number of sort work areas to be allocated by PROC SORT.[5]

  Operating systems: OS and VSE

DIAG
: passes the DIAG parameter to the sort utility. For those sort utilities that support this option, additional diagnostic information is produced in the event of a sort failure.[6]

  Operating systems: CMS, OS, and VSE

SORTSIZE=*parameter*
SIZE=*parameter*
: specifies the maximum virtual storage that can be used by the system sort utility. If not specified, the default is given by the SAS system option SORTSIZE=.[7]

  Operating systems: All

BY Statement

BY *option variable option variable ...;*

Any number of variables can be specified in the BY statement. A BY statement **must** be used with PROC SORT.

As described above, SORT first arranges the observations in the order of the first variable in the BY statement; then it sorts observations with a given value of the first variable by the second variable, and so on.

The following option can be specified in the BY statement:

DESCENDING
: sorts variables in descending order. Use the keyword DESCENDING before the name of each variable in the BY statement whose values are to be in descending order. For example, these statements:

```
PROC SORT DATA=RANDOM;
   BY DESCENDING SIZE AGE;
```

sort the RANDOM data set first by descending order of SIZE values, then by ascending order of AGE values. To sort by descending AGE values, the DESCENDING keyword would have to precede the AGE variable name.

The DESCENDING option has the same effect as the PROC SORT statement REVERSE option, except that DESCENDING can be used for both numeric and character variables. If both are specified, however, character variables are sorted in reverse descending sequence, which is identical to ascending order.

The BY statement is also discussed in "SAS Statements Used in the PROC Step."

Output Data Set

If an OUT= option appears in the PROC SORT statement, a new data set is created containing the sorted observations. Otherwise, the original data set is sorted and the sorted observations replace the original values **after** the procedure is executed without errors. As always when a data set is replaced, there must be space in the data library for a second copy of the old data set when PROC SORT is invoked.[8]

Usage Notes

Multiple sorts When you know that you need to sort a data set for several purposes, you can sometimes plan ahead and reduce the number of sorts needed. For example, suppose you want to run a procedure for the same data set twice, once with the BY statement

```
BY STATE;
```

and again with the BY statement:

```
BY STATE CITY;
```

You can sort the data set once and use it for both these runs if you use the statements

```
PROC SORT;
   BY STATE CITY;
```

It does not matter for the BY STATE; run that the data set is sorted by both state and city. If the first statement above were BY CITY;, you would need to sort the data twice.

Sorting large data sets Occasionally, a data set is too large for the sort utility to handle. If your installation uses an IBM or equivalent system sort utility, PROC SORT prints a message explaining the situation when this happens.[9]

EXAMPLE

Sorting Information

In the example below, a list of names, telephone extensions, and room numbers is sorted two different ways. PROC SORT first sorts the data alphabetically by name, then by room number and name.

```
DATA PHONES;
   INPUT NAME $ PHONE ROOM;
CARDS;
   REBECCAH 424 112
   CAROL 450 112
   LOUISE 409 110
   GINA 474 110
   MIMI 410 109
   ALICE 411 106
   BRENDA 414 105
   DAVID 438 141
   BETTY 464 141
   HOLLY 466 140
   GRETEL 465 140
PROC SORT OUT=LIST1;
   BY NAME;
PROC PRINT;
   TITLE 'NAME ORDER';
PROC SORT OUT=LIST2;
   BY ROOM NAME;
PROC PRINT;
   TITLE 'ROOM NUMBER AND NAME ORDER';
```

Output 46.1 Data Set Sorted in Name Order

NAME ORDER 1

OBS	NAME	PHONE	ROOM
1	ALICE	411	106
2	BETTY	464	141
3	BRENDA	414	105
4	CAROL	450	112
5	DAVID	438	141
6	GINA	474	110
7	GRETEL	465	140
8	HOLLY	466	140
9	LOUISE	409	110
10	MIMI	410	109
11	REBECCAH	424	112

Output 46.2 Data Set Sorted by Room Number and by Name

ROOM NUMBER AND NAME ORDER 2

OBS	NAME	PHONE	ROOM
1	BRENDA	414	105
2	ALICE	411	106
3	MIMI	410	109
4	GINA	474	110
5	LOUISE	409	110
6	CAROL	450	112
7	REBECCAH	424	112
8	GRETEL	465	140
9	HOLLY	466	140
10	BETTY	464	141
11	DAVID	438	141

NOTES

1. **AOS/VS, PRIMOS, and VMS:** The BEST sort is used unless HOST or SAS is specified.

CMS, OS, and VSE: Unless the SORTPGM=SAS option is in effect, PROC SORT calls a system sort utility to sort data sets.

2. **AOS/VS, PRIMOS, and VMS:** SORT uses the ASCII collating sequence when it compares character values. From smallest to largest, this sequence is

```
blank!"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`
abcdefghijklmnopqrstuvwxyz{|} ~
```

CMS, OS, and VSE: SORT uses the EBCDIC collating sequence (standard for IBM 360/370) when it compares character values. From smallest to largest values, this sequence is

```
blank¢.<(+ | &!$*);¬-/¦ ,%_>?`:#@'="
abcdefghijklmnopqr ~stuvwxyz {ABCDEFGHI}J
KLMNOPQR\STUVWXYZ0123456789
```

3. **VSE:** Under VSE, LIST has the same effect as the MESSAGE option.

4. **CMS:** When PROC SORT abends under CMS, you receive an error message indicating that there was insufficient virtual storage. Under CMS, the default LEAVE= value for SORT is 64000, unless SORTSIZE=MAX is specified.

OS: When PROC SORT abends under OS, you receive system completion code 80A. Under OS, the default LEAVE= value for SORT is 16000, unless SORTSIZE=MAX is specified.

5. **OS:** Under OS, the SORTWKNO= option specifies the minimum number of sort work areas if the SAS system option NODYNALLOC is in effect. The value of *number* can be 0-6. A specification of 0 causes no sort work areas to be allocated and the sort to proceed without them. The system sort utility in use must support in-core sorting without sort work areas. If you are not running SAS under OS, a specification of 0 permits the sort attempt to proceed even if there are no sort work areas allocated in the job control; if any other value (1 to 6) is specified, SAS checks that sort work areas have been allocated in the job control language.

If the SAS system option DYNALLOC is specified, the SORTWKNO= specification is passed to the system sort utility.

The SORTWKNO= option is ignored if a SORTWK01 DD statement is used. The default for this option is given by the SAS system option SORTWKNO=.

VSE: Under VSE, the value of *number* can be 0-6. If a number other than 0 is specified, there must be DLBL and EXTENT statements for each file, with DLBL names SORTWK1 to SORTWK*n*, either in the JCL or in partition standard labels. If 0 is specified, the sort utility does an in-core sort.

If SORTWKNO= is not specified, the value of the SAS system option SORTWKNO= is used. If the system option is set to a missing value, PROC SORT checks for a DLBL statement for SORTWK1. If one is present, PROC SORT sets SORTWKNO= to 1; otherwise, it is set to 0 and an in-core sort is done.

6. **VSE:** VSE always supplies this diagnostic information.

7. **OS:** When SORTSIZE=SIZE is specified, the sort executes with the amount of free space in the region minus the LEAVE= option value.

When SORTSIZE=*nnnnn* is specified, *nnnnn* bytes of memory are passed to the sort utility.

When SORTSIZE=*n*K is specified, *n*K bytes of memory are passed to the sort utility.

When SORTSIZE=MAX is specified, the parameter MAX is passed to the sort utility. This causes the sort utility to "size" itself. Note that not all sort utilities support the MAX specification.

When SORTSIZE=0 is specified, a value of zero is passed to the sort utility. In this case, most sort utility programs use a value assigned by the installation at the time of the sort utility's generation/installation.

VSE: The SORTSIZE= specifications of MAX, and 0 are invalid under VSE.

When SORTSIZE=SIZE is specified, the sort utility uses all the free space in the program area.

When SORTSIZE=*nnnnn* or *n*K is specified, the sort utility uses only nnnnn or nK bytes of the free space in the program area (unless the free space is less than *nnnnn* or *n*K). Normally, you should use or default to SORTSIZE=SIZE unless you have reason to restrict the sort utility to less memory.

8. **CMS:** If there is not enough space on the output disk to create the OUT= data set, SORT will not continue. You will receive an error message explaining that there is not enough space available on the disk to sort the data set.

9. **CMS:** To increase the sort utility's capacity, you must take the specific action required by the particular sort utility that you are using. If you have specified the SAS system option SORTPGM=SAS, you must define and format a larger temporary disk. If you are using SyncSort, you must contact your computing center staff to increase the sort work area allocation made internally by SYNCSORT. For other sort utilities, contact your computing center staff.

OS: Under OS, you can increase the sort's capacity by adding a SORT parameter to the EXEC statement in your JCL:

```
// EXEC SAS,SORT=s
```

The *s* value to use depends on what kind of disk drive the system sort uses for its work files at your installation; consult your computing center staff to find out what disk is used.

The formula for calculating the *s* value is

$$s = n*v/d$$

where

n	number of observations in the data set
v	number of variables in the data set
d	depends on the sort work device type; see the table below.

SORT WORK DEVICE TYPE	*d*
2311	3,500
2314/2319	35,000
3330/3330V/3330-1	57,000
3340/3344	24,000
3350	135,000
3375	97,000
3380	164,000

These formulas are based on the assumption that all the variables are numeric. If your data set includes long character variables, the SORT value must be larger. For safety, you should use an *s* value 20-50% larger than the values given by the formulas.

For example, suppose you want to sort a data set consisting of 5000 observations and 200 variables. Your installation uses 3330-type disks for the sort work space. You calculate the SORT value as

```
s=n*v/57000
 =5000*200/57000
 =17.5
 =18
```

Adding 50% to 18 gives an *s* value of 27, so the SORT EXEC specification is SORT=27. For very large sorts, it may be necessary to use tape SORTWK*nn* areas.

VSE: To increase the sort utility's capacity, you must either increase the size of the SORTWK1 file or allocate additional sort work files. You may need to consult with your computing center staff to accomplish this. If you increase the size of the SORTWK1 file by allocating more than one extent to it, you must specify (or default to) SORTWKNO=1 when you invoke PROC SORT. If you allocate additional sort/work files, these must have DLBL names SORTWK2, SORTWK3 ... SORTWK6. You must also use the SORTWKNO= option to tell the sort utiltity how many sort files are available. If, for example, you allocated a SORTWK2 and a SORTWK3 file, but specified SORTWKNO=2, only SORTWK1 and SORTWK2 would be used.

Chapter 47

The SOURCE Procedure

Operating systems: OS and VSE

ABSTRACT

The SOURCE procedure provides an easy way to back up and process library data sets. Use the SOURCE procedure to:

- produce control data sets for other library processing utilities
- print or unload library members[1]
- process library directories to produce reports on the library contents.[2]

INTRODUCTION

The SOURCE procedure can:

- print the contents of an entire library on the SAS log
- process only the directory of a library to produce input for utility, SAS, or other programs

---

OS LIBRARIAN is a registered trademark of Applied Data Research, Inc., Princeton, NJ, USA.

- route the members of a library to other programs for processing[3]
- create a sequential, or unloaded, version of the library's directory records
- construct an unloaded data set from a library.[4]

A source library can be copied into a sequential tape or disk data set to create a backup or a transportable copy of the source data. This copy is called an *unloaded data set* and consists of 80-byte records containing the source data and control information needed to restore the source back to its original organization. When an unloaded data set is restored with the proper utility to a device that will support the data in their original form, the data are reconstructed, or *loaded*.

An advantage of having an unloaded data set is that one or more members can be retrieved without reloading the entire library. Refer to the RELOAD member in the SAS sample library for an example that shows how to do this.

PROC SOURCE has several advantages over IBM's IEBPTPCH (OS) and SSERV (VSE) utilities. With PROC SOURCE you can:

- print members in alphabetical order
- select members by specifying a wildcard or range
- print the number of records in each member
- choose to print each member on a new page
- produce an unloaded, portable version of the OS or VSE library.

Consult the books listed in the **REFERENCES** section of this chapter for more information on the libraries that can be processed with PROC SOURCE.

SPECIFICATIONS

The following statements are used with PROC SOURCE:

PROC SOURCE *options*;
  SELECT *member ...*;
  EXCLUDE *member ...*;
  FIRST *'model control statement' ...*;
  LAST *'model control statement' ...*;
  BEFORE *'model control statement' options ...*;
  AFTER *'model control statement' options ...*;

The *model control statements* in the FIRST, LAST, BEFORE, and AFTER statements are usually either utility or job control statements, depending on the destination given by the OUTDD= option in the PROC SOURCE statement.

PROC SOURCE Statement

PROC SOURCE *options;*

The following options are used in the PROC SOURCE statement.

DIRDD=*fileref* gives the *fileref* of the output data set to which PROC SOURCE writes a sequential, unloaded form of the PDS or SSL directory.[5] Each directory record is written into one 80-byte record, left aligned, and padded on the right with blanks. The fileref must match the reference name used in control language statements describing the output data set.

Operating systems: OS and VSE

INBLK=*blocksize* gives the *blocksize* to use when reading the input library. This option can be used to override the data set's (DSCB) blocksize when it has been incorrectly altered or destroyed, and thus permits the contents of the library to be retrieved correctly. The INBLK= specification overrides any control language specification of blocksize.

Operating system: OS

INDD=*fileref* gives the *fileref* specified in the control language describing the input library. If the INDD= option is not specified, the default fileref is SOURCE.

Operating system: OS

MAXIOERROR=*number* specifies the maximum *number* of I/O errors to allow before terminating. Normally, PROC SOURCE detects, issues a warning message about, and then ignores I/O errors that occur while reading the library members. When the number of errors specified by MAXIOERROR= has occurred, however, PROC SOURCE assumes that the library is unreadable and stops. The default MAXIOERROR= value is 50.

Operating system: OS

NOALIAS treats aliases as main member names. Therefore, PROC SOURCE does not generate ./ ALIAS cards or alias BEFORE and AFTER cards.

Operating system: OS

NODATA indicates that you do not want to read or write (to the OUTDD= file) the members in the input library. In other words, PROC SOURCE produces only control statements and a list of the member names and does not output the member source records.[6] NODATA is particularly useful when you want to process only the directory of a library.

Operating systems: OS and VSE

NOPRINT indicates that you do not want to print the list of member names and record counts. (These listings are produced even when the PRINT option is **not** specified.) The NOPRINT option is ignored when PRINT is specified.

Operating systems: OS and VSE

NOSUMMARY indicates that you do not want to print the member summary. The NOSUMMARY option is ignored when the NODATA, NOPRINT, or PRINT option is specified.

Operating systems: OS and VSE

NOTSORTED causes PROC SOURCE to process library members in the order in which they:

- appear in SELECT statements
- remain after EXCLUDE statements.

Normally, PROC SOURCE processes (that is, unloads, prints, and so on) the library members in alphabetical order by member name.

Operating systems: OS and VSE

NULL indicates that null members (library members with no records, just an immediate end-of-file) should be processed. Such members occasionally appear in source libraries, but they are not normally unloaded because IEBUPDTE and most other source library maintenance utilities do not create null members. If the source library maintenance utility you are using can properly recognize and create a null member, specify this option and provide the appropriate BEFORE (and possibly, AFTER) statements.

Operating system: OS

OUTBLK=*blocksize* specifies the *blocksize* of the OUTDD= data set.[7]

If you do not use the OUTBLK= option, the default is the SAS system option FILEBLKSIZE(*device*) value that is appropriate for your output device. The file always has a record length of 80 and a fixed block record format.

Operating systems: OS and VSE

OUTDD=*fileref* specifies the *fileref* of the output file to which PROC SOURCE writes the unloaded (sequential) form of the input library and any records that FIRST, LAST, BEFORE and AFTER statements generate. The fileref must match the reference name used in control language describing the data set.

Operating systems: OS and VSE

PAGE begins the listing of each member on a new page.

Operating systems: OS and VSE

PRINT prints the entire OS library or VSE sublibrary. The PRINT option is ignored when NODATA is specified.

Operating systems: OS and VSE

SEARCH indicates that a LIBDEF statement specifies a SEARCH chain that tells which source statement libraries to search for desired member(s).

The SELECT and EXCLUDE colon (:) and hyphen (-) notation cannot be used with the SEARCH option.

When you do not use the SEARCH option, the library specified in the *FROM*= operand in a LIBDEF statement is used for input.

Operating system: VSE

SUBLIB=*oneletter* specifies the sublibrary to use when accessing the source statement library. Only one sublibrary can be given. The default value for SUBLIB= is A.

Operating system: VSE

SELECT Statement

SELECT *member ...;*

Use the SELECT statement to process only certain library members, rather than all members (the default). When you use the SELECT statement, only the members specified are processed. You can use any number of SELECT statements.

Use the colon (:) notation to indicate that you want all members beginning with the characters preceding the colon. (See the example below.)

You can include an alphabetic range of names in the SELECT statement by joining two names with a hyphen (-). The two hyphenated members and all members in between are processed. For example, if a library contains members called BROWN, GREEN, GRAY, RED, and YELLOW, and you want to process the first four members, use this SELECT statement:

```
SELECT BROWN-RED;
```

The colon (:) and hyphen (-) notation can be used together.[8] For example:

```
SELECT BR:-GR: RED;
```

is equivalent to the previous SELECT statement.

EXCLUDE Statement

EXCLUDE *member ...;*

Use the EXCLUDE statement to exclude certain members from processing. When you use the EXCLUDE statement, all members except those specified are processed. You can use any number of EXCLUDE statements.

Use the colon (:) notation to indicate that you want to exclude all members with names beginning with the characters preceding the colon.

You can include an alphabetic range of names in the EXCLUDE statement by joining two names with a hyphen. The two hyphenated members and all members in between are excluded from processing. (See the example in the SELECT statement description.)

The colon and hyphen notation can be used together. (See the SELECT example above).[9]

It is convenient to use SELECT and EXCLUDE statements together when you want to select many members using colon or hyphen notation but exclude a few of the members selected. For example, if there are 200 members called SMC1-SMC200, and you want to copy all of them except SMC30-SMC34, use these statements:

```
SELECT SMC:;
EXCLUDE SMC30-SMC34;
```

When you use both EXCLUDE and SELECT statements, the EXCLUDE statements should specify only members that are specified by the SELECT statements. However, excluding unselected members has no effect other than to generate warning messages.

FIRST Statement

FIRST *'model control statement' ...;*

The FIRST statement generates initial control statements that either invoke a utility program or that are needed only once. The *model control statement* specified is reproduced, left-aligned, on a record that precedes all members in the unloaded data set. You can use any number of FIRST statements. One FIRST statement can specify any number of model control statements, with each model control statement generating a record.

LAST Statement

LAST *'model control statement'* ...;

The LAST statement generates final control statements that either terminate a utility program or that are needed only once. The *model control statement* specified is reproduced, left-aligned, on a record that follows all members in the unloaded data set. You can use any number of LAST statements. One LAST statement can specify any number of model control statements, with each model control statement generating a record.

BEFORE Statement

BEFORE *'model control statement'* options ...;

The BEFORE statement generates utility control statements before each member. You can use any number of BEFORE statements. One BEFORE statement can specify any number of model control statements and options. Each *model control statement* specified is reproduced, left-aligned, on a record that precedes each member in the unloaded data set.

By default, PROC SOURCE generates certain utility statements before each member of an unloaded data set.[10] Use the BEFORE (and AFTER) statement to override the default and generate control statements for other utility programs.

Options for the BEFORE and AFTER statements are the same. A list of these options follows the AFTER statement description.

AFTER Statement

AFTER *'model control statement'* options ...;

The AFTER statement generates utility control statements after each member. You can use any number of AFTER statements. One AFTER statement can specify any number of control statements and options. Each *model control statement* specified is reproduced, left-aligned, on a record that follows each member in the unloaded data set.

By default, SOURCE generates control statements for certain IBM utility programs after each member of an unloaded data set. You can override the default and generate control statements for other utility programs with the AFTER statement.

The following options are used in the BEFORE and AFTER statements:

ALIAS
: indicates that a record containing the *model control statement(s)* is to be produced only for each alias defined (with the alias being placed into the record at the specified column, if any).

  Operating system: OS

column number
: specifies the beginning column number of a field containing the member name in records generated by BEFORE and AFTER statements.[11] The name is left-

aligned in the field and padded on the right with blanks.

Operating systems: OS and VSE

FULL indicates that the full member name (sublibrary and member name, separated by a period) is to be placed in the column number specified. By default, the member name without the sublibrary and separator period is printed.

Operating systems: VSE

NOBLANK eliminates the blank(s) between the end of the member name or alias and any text that follows. The example below shows a record with a member name preceding the text without the NOBLANK option:

name ,text text text

The same record with NOBLANK specified looks like this:

name,text text text

Operating systems: OS and VSE

RIGHT right-aligns the member name, or library and member name, in the specified field. By default, the names are left-aligned in an 8-column field.[12]

The example below shows a record with the member name preceding the text without the RIGHT option.

name ,text text text

The same record with the RIGHT option specified looks like this:

name,text text text

Operating systems: OS and VSE

DETAILS

Printed Output

PROC SOURCE prints:

1. the entire library, if the PRINT option is specified
2. a listing of the member names in the library (unless you specify NOPRINT)
3. the number of records for each member (unless you specify NOPRINT or NODATA)
4. a summary of the attributes and contents of the library
5. the SSL from which the member was retrieved when the SEARCH option is specified.

Even when PRINT is not specified, some records may still be printed. The signal "NAME:" or "ENTRY:" or "AUTHOR:" beginning in column 5 of a record in the

library starts the printing; the signal "END" beginning in column 5 stops it. If you do not want this subset of records printed, specify the NOSUMMARY option.

EXAMPLES

The example output has been edited to conserve space.

Printing Selected Members from an OS PDS: Example 1

The job below prints, starting with the member *FORTHC* through the member named *FORTHCL*.

```
//jobname JOB account,user
// EXEC SAS
//SOURCE DD DSN=ABCD.DEF.PROCLIB,DISP=SHR
PROC SOURCE PRINT;
   SELECT FORTHC-FORTHCL;
```

Output 47.1 Selecting Members from a Source Statement Library

```
        S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXMP1 STEP SASBASE  PROC

NOTE: THE JOB SRCEXMP1 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.09 SECONDS.
1            PROC SOURCE PRINT;
2                SELECT FORTHC-FORTHCL;
3                TITLE 'MEMBERS FORTHC THROUGH FORTHCL';
```

```
2       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXMP1 STEP SASBASE  PROC
   FORTHC  ❷
               //FORT   EXEC  PGM=IEKAA00
               //SYSPRINT DD  SYSOUT=$
               //SYSPUNCH DD  SYSOUT=B
               //SYSLIN   DD  DSNAME=&LOADSET,UNIT=VIODA,DISP=(MOD,PASS),
  ❸            //             SPACE=(400,(200,50),RLSE),DCB=BLKSIZE=3120
   5-RECORDS
   FORTHCL ❷
               //FORT   EXEC  PGM=IEKAA00
               //SYSPRINT DD  SYSOUT=$
               //SYSPUNCH DD  SYSOUT=B
               //SYSLIN   DD  DSNAME=&LOADSET,UNIT=VIODA,DISP=(MOD,PASS),
               //             SPACE=(400,(200,50),RLSE),DCB=BLKSIZE=3120
               //LKED EXEC PGM=IEWL,PARM=(MAP,LET,LIST),COND=(4,LT,FORT)
               //SYSLIB   DD  DSNAME=ABC1.FORTLIB,DISP=SHR
               //SYSPRINT DD  SYSOUT=$
               //SYSLMOD  DD  DSNAME=&GOSET(MAIN),UNIT=VIODA,DISP=(,PASS),
               //             SPACE=(3072,(30,10,1),RLSE)
               //SYSLIN   DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)
               //           DD  DDNAME=SYSIN
  ❸            //SYSUT1 DD DSNAME=&SYSUT1,UNIT=VIODA,SPACE=(1024,(200,20)),SEP=SYSLMOD
   13-RECORDS

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

     ❹    SOURCE LIBRARY DATA SET: INDD=SOURCE,BLKSIZE=3200,
          DSNAME=ABCD.DEF.PROCLIB,VOL=SER=XXX111

       89   MEMBERS DEFINED IN SOURCE LIBRARY.

        0   ALIASES DEFINED IN SOURCE LIBRARY.
```

(continued on next page)

(continued from previous page)

```
      35   DIRECTORY BLOCKS ALLOCATED.

      14   DIRECTORY BLOCKS USED.

       2   MEMBERS SELECTED.

      18   RECORDS READ FROM SOURCE LIBRARY.
```

```
3       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXMP1 STEP SASBASE  PROC

NOTE: THE PROCEDURE SOURCE USED 0.12 SECONDS AND 352K.
NOTE: SAS USED 352K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Printing Selected Members from a VSE SSL: Example 2

The job below prints selected members from the *SASXMPL* source statement library. Members *REPORT* through *REPORT9* are printed. (The records are not shown in the output.)

```
// JOB jobname
// ASSGN SYSIPT,00C
// ASSGN SYS011,01E
// ASSGN SYS006,00E
// DLBL WORK000,'data set name'
// EXTENT SYS000,volser,1,0,begtrks,numtrks
// ASSGN SYS000,DISK,VOL=volser,SHR
// LIBDEF SL,FROM=SASXMPL,TEMP
// EXEC SASVSE,SIZE=(SASVSE,40K)
PROC SOURCE PRINT SUBLIB=S ;
   SELECT REPORT-REPORT9;
/*
/&
```

Output 47.2 Printing Selected Members from a VSE SSL

```
        S A S   L O G   VSE SAS 5.XX        VSE 1.3.0 JOB SAMPLE

NOTE: THE JOB SAMPLE HAS BEEN RUN UNDER RELEASE 5.XX OF SAS

NOTE: CPUID   VERSION = FF  SERIAL = 123456  MODEL = 7890 .

NOTE: NO OPTIONS SPECIFIED.

NOTE: THE INITIALIZATION PHASE USED 0.08 SECONDS.
1             PROC SOURCE PRINT SUBLIB=S ;
2             SELECT REPORT-REPORT9 ;
```

```
2       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

S.REPORT
               /*****************************************************************/
❷              /*          S A S   S A M P L E   L I B R A R Y                  */
               /*                                                               */

... NOT ALL RECORDS SHOWN ...                                                   */

                    / @C +5   WEIGHT=
                    / @C +5   N=
                    //@C +5   AGE=
                    / @C +5   SEX=;

❸                 C+21;  IF C<113 THEN RETURN;  IF NOT EOF THEN PUT _PAGE_; C=1;
  290-RECORDS
```

```
7       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

S.REPORT2
               /*****************************************************************/
❷              /*          S A S   S A M P L E   L I B R A R Y                  */
               /*                                                               */

... NOT ALL RECORDS SHOWN ...                                                   */

                  L + 1; IF L <=25 THEN RETURN;
                  C + 30; L = 1; IF C < 62 THEN RETURN;
❸                 PUT _PAGE_; C = 1;
  103-RECORDS
```

```
9       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

S.REPORT3
               /*****************************************************************/
❷              /*          S A S   S A M P L E   L I B R A R Y                  */
               /*                                                               */

... NOT ALL RECORDS SHOWN ...                                                   */

                        '    MISC. PROMOTION' // ' TOTAL PROMOTION' ////
                       'PRODUCT CONTRIBUTION ';

                RETURN;
  116-RECORDS
```

```
11      S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

S.REPORT4
               /*****************************************************************/
               /*          S A S   S A M P L E   L I B R A R Y                  */
               /*                                                               */

... NOT ALL RECORDS SHOWN ...                                                   */

                  TITLE1 FIRST QUARTER SALES FOR SUNY STEREO INC.;
                  TITLE2 IN DOLLARS BY CITY;

  54-RECORDS
```

```
12      S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

S.REPORT5
               /*****************************************************************/
               /*          S A S   S A M P L E   L I B R A R Y                  */
               /*                                                               */

... NOT ALL RECORDS SHOWN ...                                                   */

                        ; END;
                IF NOT EOF THEN PUT _PAGE_;
  74-RECORDS

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

(continued on next page)

(continued from previous page)

```
          SOURCE LIBRARY DATA SET: FILENAME=SASXMPL,VOLSER=VSEXYZ ❺
❹     5   MEMBERS SELECTED.
      637   RECORDS READ FROM SOURCE LIBRARY.
```

```
14      S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

NOTE: THE PROCEDURE SOURCE USED 2.96 SECONDS AND 354K.
NOTE: SAS USED 354K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Printing an Entire OS Library: Example 3

The job below prints the entire contents of an OS source library.

```
//jobname JOB account,user
// EXEC SAS
//MYLIB DD DSN=MY.SOURCE.LIBRARY,DISP=SHR
PROC SOURCE INDD=MYLIB PRINT;
```

Output 47.3 Printing the Entire Contents of an OS Source Library

```
        S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC

NOTE: THE JOB SRCEXM3 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.08 SECONDS.
1            PROC SOURCE INDD=MYLIB PRINT;
2               TITLE 'PRINT THE SOURCE STATEMENT LIBRARY';
```

```
2       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC
   $PUTMATU
               %INCLUDE MACPORT;
               $PUTMAT: PROC(MATPTR,ROW1,ROW2,COL1,COL2,NDIM,SHAPE,
                        WIDTH2,FORMAT,NSKIP2,
...NOT ALL RECORDS SHOWN...
               END;
               RETURN;

   101-RECORDS
```

```
4       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC
   $SFXL
               %INCLUDE MACPORT;
               $SFXL: PROC(FIELD,CDFLT,POS,LENGTH,STRING) PORTPROC;
                    LENGTH=FIELDN;
                    XLPTR=FIELDADDR;
...NOT ALL RECORDS SHOWN...
                    $PL IF(SFDEBUG) DEL(/) I(LENGTH) P(XLPTR)
                        C(SUBSTR(XLPTR->XLSTR,1,LENGTH));
                    RETURN;
               END $SFXL;

   82-RECORDS
```

(continued on next page)

(continued from previous page)

```
   BATCHSAS
               //TESTPROC JOB (,C140),WHITTLE,NOTIFY=,MSGLEVEL=(2,0),MSGCLASS=Z,
               //    TIME=(,10)
               /*JOBPARM FETCH
...NOT ALL RECORDS SHOWN...
               PROC SORT DATA=A.TPUTMTL;
                  BY Y;
               PROC PRINT DATA=A.TPUTMTL;
   20-RECORDS
```

```
6       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC

   FORMS
               %INCLUDE MACPORT;
               FORMS:  PROC
                           OPTIONS(MAIN) REORDER /*IBMONLY*/
                           ;
                    IF IVAL2-=IVAL THEN CALL XLOGA
...NOT ALL RECORDS SHOWN...
                    RETURN(IVAL2);
                    END PARMSET;
              0     END FORMS;
   534-RECORDS

   HSWMORG
               %INCLUDE MACPORT;
               HSWMORG: PROC PORTPROC RETURNS(PTR);
               DCL  TOD CHAR(50) STATIC INIT('HSWMORG TODSTAMP: 09/28/84 14:49:39');
               DCL  ACTMAP   (      75) FXS STATIC INIT(
...NOT ALL RECORDS SHOWN...
               %DECLARE INSRTSIZE  CHAR; %INSRTSIZE ='      59  ';
               %INCLUDE GRAMFUNC;
                    END HSWMORG;
   198-RECORDS

   HSWMORT
               %INCLUDE MACPORT;
               HSWMORT: PROC PORTMAIN;
...NOT ALL RECORDS SHOWN...
                    CALL XPSKIP(2);
                    END NEWPAGE;
                    END HSWMORT;
   293-RECORDS
```

```
24      S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC

   HWPUTMG
               %INCLUDE MACPORT;
               HWPUTMG: PROC PORTPROC RETURNS(PTR);
               DCL  TOD CHAR(50) STATIC INIT('HWPUTMG TODSTAMP: 10/19/84 16:31:28');
               DCL  ACTMAP   (      81) FXS STATIC INIT(
...NOT ALL RECORDS SHOWN...

               %DECLARE INSRTSIZE  CHAR; %INSRTSIZE ='      82  ';
               %INCLUDE GRAMFUNC;
                    END HWPUTMG;
   234-RECORDS

   HWPUTMT
               %INCLUDE MACPORT;
               HWPUTMT: PROC PORTMAIN;
...NOT ALL RECORDS SHOWN...
               CALL XEXIT(XEXITNORMAL,0);
               END;
   329-RECORDS
```

```
34      S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC

   OHWPUTMT
               %INCLUDE MACPORT;
               HWPUTMT: PROC PORTMAIN;
...NOT ALL RECORDS SHOWN...
               CALL XEXIT(XEXITNORMAL,0);
               END;
   328-RECORDS
```

(continued on next page)

(continued from previous page)

```
   PRECHANG
               %INCLUDE MACPORT;
               HWPUTMT: PROC PORTMAIN;
...NOT ALL RECORDS SHOWN...
               CALL XEXIT(XEXITNORMAL,0);
               END;
   328-RECORDS

   TSTPUTMT
              //TESTPROC JOB (,C140),WHITTLE,NOTIFY=,MSGLEVEL=(2,0),MSGCLASS=Z,
              //    TIME=(,10)
              /*JOBPARM FETCH
...NOT ALL RECORDS SHOWN...
                    COR='VARIABLE NAME'
                    SIG OBS
                    DOU
                    UND TRA;
                  BY X Y;
   172-RECORDS

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

           SOURCE LIBRARY DATA SET: INDD=MYLIB,BLKSIZE=6160,
           DSNAME=SASABC.PGV5.SOURCE,VOL=SER=XYZ802

      12   MEMBERS DEFINED IN SOURCE LIBRARY.

       0   ALIASES DEFINED IN SOURCE LIBRARY.

      11   DIRECTORY BLOCKS ALLOCATED.

       2   DIRECTORY BLOCKS USED.

      12   MEMBERS SELECTED.

    2623   RECORDS READ FROM SOURCE LIBRARY.
```

```
49      S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM3  STEP SASBASE  PROC

NOTE: THE PROCEDURE SOURCE USED 0.35 SECONDS AND 356K.
NOTE: SAS USED 356K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Printing an Entire VSE Library: Example 4

The job below prints the entire contents of the *S* sublibrary contained in the *SASXMPL* SSL.

```
// JOB jobname
// ASSGN SYSIPT,00C
// ASSGN SYS011,01E
// ASSGN SYS006,00E
// DLBL WORK000,'data set name'
// EXTENT SYS000,volser,1,0,begtrks,numtrks
// ASSGN SYS000,DISK,VOL=volser,SHR
// LIBDEF SL,FROM=SASXMPL
// EXEC SASVSE,SIZE=(SASVSE,40K)
PROC SOURCE PRINT SUBLIB=S ;
/*
/&
```

Output 47.4 Printing an Entire Sublibrary

```
        S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE
NOTE: THE JOB SAMPLE HAS BEEN RUN UNDER RELEASE 5.XX OF SAS
NOTE: CPUID   VERSION = FF  SERIAL = 123456  MODEL = 7890 .
NOTE: NO OPTIONS SPECIFIED.
1             PROC SOURCE PRINT SUBLIB=S ;
```

```
2       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE
S.ALPHA
                /*************************************************************/
                /*          S A S   S A M P L E   L I B R A R Y              */
                /*                                                           */
..NOT ALL RECORDS SHOWN...
                  T6=T5-T4; T7=T5#/(T4<>1E-12);
                  STATSTAT=STATSTAT//(T1||T2||T3||T4||T5||T6||T7);
                  RETURN;
   109-RECORDS
```

```
4       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE
S.ANOVA
                /*************************************************************/
                /*          S A S   S A M P L E   L I B R A R Y              */
                /*                                                           */
..NOT ALL RECORDS SHOWN...
                  TEST H=S E=S*REP;
                  TEST H=S*F E=S*REP*F;
                  TEST H=S*CA S*F*CA E=S*CA*REP(F);
   67-RECORDS

S.ANOVA2
                /*************************************************************/
                /*          S A S   S A M P L E   L I B R A R Y              */
                /*                                                           */
..NOT ALL RECORDS SHOWN...
                  TEST H=HARVEST E=HARVEST*REP;
                  TEST H=HARVEST*VARIETY;
```

```
6       S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE
   45-RECORDS

S.ANOVA3
                /*************************************************************/
                /*          S A S   S A M P L E   L I B R A R Y              */
..NOT ALL RECORDS SHOWN...
                  TEST H=HARVEST E=HARVEST*REP;
               493=JAN01  494=FEB01  495=MAR01  496=APR01  497=MAY01  498=JUN01
               499=JUL01  500=AUG01  501=SEP01  502=OCT01  503=NOV01  504=DEC01 ;
   103-RECORDS
```

```
251     S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          SOURCE LIBRARY DATA SET: FILENAME=SASXMPL,VOLSER=XXXYYY

     147  MEMBERS SELECTED.

   13729  RECORDS READ FROM SOURCE LIBRARY.
```

```
252     S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPLE

NOTE: THE PROCEDURE SOURCE USED 72.81 SECONDS AND 338K.
NOTE: SAS USED 338K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Unloading an OS Library to Tape: Example 5

The job below produces an unloaded data set on tape of the input source library.

```
//jobname JOB account,user
// EXEC SAS
//IN DD DISP=SHR,DSN=ABCDEF.OLD.SOURCE
//TAPE DD DSN=ABCDEF.PGVS.SOURCE,DISP=(NEW,KEEP),UNIT=TAPE
PROC SOURCE NOPRINT INDD=IN OUTDD=TAPE;
```

Output 47.5 Producing an Unloaded Source Library on Tape

```
        S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM5  STEP SASTEST  PROC

NOTE: THE JOB SRCEXM5 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS AT SAS INSTITUTE INC. (TEST) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.12 SECONDS.
1            PROC SOURCE NOPRINT INDD=IN OUTDD=TAPE;


           SOURCE LIBRARY DATA SET: INDD=IN,BLKSIZE=6160,
           DSNAME=ABCDEF.PGV5.SOURCE,VOL=SER=XYZ802

      12   MEMBERS DEFINED IN SOURCE LIBRARY.

       0   ALIASES DEFINED IN SOURCE LIBRARY.

      11   DIRECTORY BLOCKS ALLOCATED.

       2   DIRECTORY BLOCKS USED.

      12   MEMBERS SELECTED.

    2623   RECORDS READ FROM SOURCE LIBRARY.

    2635   RECORDS WRITTEN TO OUTDD=TAPE,
           LABEL=(1,SL),DISP=NEW,DCB=(DEN=4,BLKSIZE=8000),
           DSNAME=ABCDEF.PGV5.SOURCE,VOL=SER=DEF123

NOTE: THE PROCEDURE SOURCE USED 0.30 SECONDS AND 376K.
NOTE: SAS USED 376K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Creating Input for the VSE CORGZ Utility Program: Example 6

This example copies all assembler macros with names starting with A, B, or C from the SASSSL library to the TOSSL library. The PROC SOURCE step retrieves the names of all sublibrary *E* books (assembler macros) beginning with the letters A through C, from the SASSSL library. The following default control cards are written to the OUTDD= file:

- CATALS E.*membername*
  BKEND card
- BKEND card

The first DATA step reads the OUTDD= file and creates the SAS data set MACLIB containing the member names. The second DATA step uses MACLIB to create input for the CORGZ utility program. The CORGZ program executes in the second job step and copies the books from SASSSL to TOSSL. (See the *SAS Companion for the VSE Operating System* for more information on using SYSPCH and SYSIPT.)

```
// JOB jobname
// ASSGN SYSIPT,00C
// ASSGN SYS011,01E
// ASSGN SYS006,00E
// DLBL WORK000,'data set name'
// EXTENT SYS000,volser,1,0,begtrks,numtrks
// ASSGN SYS000,DISK,VOL=volser,SHR
// DLBL TEMP,'data set name'
// EXTENT SYS031,volser,1,0,begtrks,numtrks
// ASSGN SYS031,DISK,VOL=volser,SHR
// DLBL SASSSL,'data set name'
// EXTENT ,volser
// DLBL TOSSL,'data set name'
// EXTENT ,volser
// LIBDEF SL,FROM=SASSSL,TO=TOSSL
  ASSGN SYSPCH,SYSLNK
// EXEC SASVSE,SIZE=(SASVSE,40K)
PROC SOURCE OUTDD=TEMP SUBLIB=E NODATA OUTBLK=6400 ;
   SELECT A:-C:  ;
DATA MACLIB ;
   INFILE TEMP RECFM=FB LRECL=80 BLKSIZE=6400 ;
      INPUT @2 XX $CHAR5. @ ;
      IF XX EQ 'CATAL' THEN DO ;
         INPUT @11 NAME $8. ;
         OUTPUT ;
         END ;
      ELSE DO ;
         INPUT ;
         END ;
         DROP XX ;

DATA COPIED ;
   SET MACLIB END=E ;
FILE SYSPCH RECFM=FB LRECL=80 BLKSIZE=80 ;
   IF _N_ = 1 THEN DO ;
      PUT @1 ' MERGE PRV,PRV ' ;
      END ;
   PUT @1 'COPYS E.' NAME ;
   IF E THEN DO ;
      PUT @1 '/*' ;
      END ;
/*
 CLOSE SYSPCH,00D
 ASSGN SYSIPT,SYSLNK
// EXEC CORGZ
/*
/&
```

Output 47.6 Selecting Assembler Macros from the SASSSL Sublibrary

```
        S A S   L O G   VSE SAS 5.XX      VSE 1.3.0 JOB SAMPL3

NOTE: THE JOB SAMPL3 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS

NOTE: CPUID   VERSION = FF  SERIAL = 123456  MODEL = 7890 .

NOTE: NO OPTIONS SPECIFIED.

1          PROC SOURCE OUTDD=TEMP SUBLIB=E NODATA OUTBLK=6400;
2               SELECT A:-C: ;


NOTE: THE NODATA OPTION HAS BEEN SPECIFIED WITH OUTDD=;
      ONLY 'FIRST', 'BEFORE', 'AFTER', AND 'LAST'
      CONTROL CARDS WILL BE WRITTEN TO OUTDD=.
E.ADCON
E.COMP

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

            SOURCE LIBRARY DATA SET: FILENAME=SASSSL,VOLSER=ABC226

       2    MEMBERS SELECTED.

       0    RECORDS READ FROM SOURCE LIBRARY.

       6    RECORDS WRITTEN TO OUTDD=TEMP    ,BLOCKSIZE= 6400
NOTE: THE PROCEDURE SOURCE USED 10.25 SECONDS AND 354K.
```

Output 47.7 Creating the SAS Data Set MACLIB

```
3          DATA MACLIB;
4          INFILE TEMP RECFM=FB LRECL=80 BLKSIZE=6400 ;
5               INPUT @2 XX $CHAR5. @ ;
6               IF XX EQ 'CATAL' THEN DO;
7                    INPUT @11 NAME $ 8.;
8                    OUTPUT;
9                    END;
10              ELSE DO;
11                   INPUT;
12                   END;
13              DROP XX;


NOTE: INFILE TEMP HAS THE FOLLOWING CHARACTERISTICS:
      DCB=(BLKSIZE=6400,LRECL=80,RECFM=FB)

NOTE: 6 LINES WERE READ FROM INFILE TEMP.
NOTE: DATA SET USER010.MACLIB HAS 2 OBSERVATIONS AND 1 VARIABLES. 678 OBS/TRK.
NOTE: THE DATA STATEMENT USED 3.87 SECONDS AND 342K.

14         PROC PRINT;
NOTE: THE PROCEDURE PRINT USED 2.77 SECONDS AND 466K AND PRINTED PAGE 1.
```

```
2       S A S   L O G   VSE SAS 5.00      VSE 1.3.0 JOB SAMPL3

NOTE: SAS USED 466K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Output 47.8 Statistics from the CORGZ Utility

```
14          DATA COPIED ;
15               SET MACLIB END=E ;
16          FILE SYSPCH RECFM=F LRECL=80 BLKSIZE=80 ;
17          IF _N_ =1 THEN DO ;
18                    PUT @1 ' MERGE PRV,PRV' ;
19                    END ;
20          PUT @1 ' COPYS E.' NAME ;
21          IF E THEN DO ;
22               PUT @1 '/*' ;
23               END ;


NOTE: FILE SYSPCH HAS THE FOLLOWING CHARACTERISTICS:
      DCB=(BLKSIZE=80,LRECL=80,RECFM=F)

NOTE: 4 LINES WERE WRITTEN TO FILE SYSPCH.
NOTE: DATA SET USER010.COPIED HAS 2 OBSERVATIONS AND 1 VARIABLES. 678 OBS/TRK.
NOTE: THE DATA STATEMENT USED 3.44 SECONDS AND 336K.

NOTE: SAS USED 466K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

```
 MERGE PRV,PRV
 COPYS E.ADCON
 COPYS E.COMP




 S T A T U S    R E P O R T                  DATE: 07/13/84 (MM/DD/YY)    TIME: 12.15 (HH.MM)          DECIMAL NUMBERS

  LIBRARIES ON FIXED     STARTING   NEXT AVAILABLE  LAST BLOCK  BLOCKS     BLOCKS     BLOCKS     ENTRIES  ACTIVE DIR
  BLOCK ARCHITECTURE     ADDRESS    ENTRY & MEMBER  ALLOCATED  ALLOCATED  ACTIVE     DELETED    & BLOCKS  ENTRIES &
  (FBA) DEVICES:        (BLOCKNO)  (BLOCKNO  BYTE)  (BLOCKNO)                                   AVAILABLE COND.LIMIT (%)


TOSSL    VOLID: VSEYYY
SOURCE-STMT  DIRECTORY    548000     548001   78      548004         5          2                  101          2   4
             LIBRARY      548005     548013           548499       495          4          4       487          0   1


 CLOSE SYSIPT,00C
```

Copying Members to an OS LIBRARIAN® Master File: Example 7

The job below produces a disk data set containing control statements that can be processed by LIBRARIAN, a proprietary product of Applied Data Research, Inc. Using this data set as input, the LIBRARIAN program copies each member of the input library (in this example, ABC2.PROCLIB) to the LIBRARIAN master file defined in the LIBRN procedure.

```
//jobname JOB account,user
// EXEC SAS
//PDS DD DISP=SHR,DSN=ABC2.PROCLIB
//SDS DD UNIT=DISK,DISP=(,PASS),DSN=&&LIBRNIN,SPACE=(TRK,(4,2))
PROC SOURCE INDD=PDS OUTDD=SDS NOALIAS NODATA;
   FIRST '-OPT NOLIST,NOEXEC,NOPC';
   BEFORE '-DLM XXXXXXXX' 6;
```

```
   BEFORE '-ADD XXXXXXXX,SEQ=/81,6,10,10/' 6 RIGHT;
   BEFORE '-AUX PDS(XXXXXXXX)' 10 NOBLANK ;
/*
//EXEC LIBRN
//PDS DD DISP=SHR,DSN=ABC2.PROCLIB
//SYSIN DD DISP=(OLD,DELETE),DSN=&&LIBRNIN
```

Output 47.9 Producing Control Statements for LIBRARIAN®

```
        S A S   L O G    OS SAS 5.XX          VS2/MVS JOB LIBRARN  STEP SASBASE  PROC

NOTE: THE JOB LIBRARN HAS BEEN RUN UNDER RELEASE 5.XX OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.09 SECONDS.
1              PROC SOURCE INDD=PDS OUTDD=SDS NOALIAS NODATA;
2                  FIRST '-OPT NOLIST,NOEXEC,NOPC';
3                  BEFORE '-DLM XXXXXXXX' 6;
4                  BEFORE '-ADD XXXXXXXX,SEQ=/81,6,10,10/' 6 RIGHT;
5                  BEFORE '-AUX PDS(XXXXXXXX)' 10 NOBLANK ;


NOTE: THE NODATA OPTION HAS BEEN SPECIFIED WITH OUTDD=;
      ONLY 'FIRST', 'BEFORE', 'AFTER', AND 'LAST'
      CONTROL CARDS WILL BE WRITTEN TO OUTDD=.
   $$$$$MX
   $$$$$M1
   $$$$$0X
   $$$$$01
...NOT ALL MEMBERS SHOWN...
   VARFSR
   VSAMBILL
   VSSECUR
   WTO

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

           SOURCE LIBRARY DATA SET: INDD=PDS,BLKSIZE=3200,
           DSNAME=SYSA.PROCLIB,VOL=SER=VOL111

     476   MEMBERS DEFINED IN SOURCE LIBRARY.

      48   ALIASES DEFINED IN SOURCE LIBRARY.

     107   DIRECTORY BLOCKS ALLOCATED.

      71   DIRECTORY BLOCKS USED.

     524   MEMBERS SELECTED.

       0   RECORDS READ FROM SOURCE LIBRARY.

    1573   RECORDS WRITTEN TO OUTDD=SDS,
           LABEL=(,SL),DISP=NEW,DCB=(BLKSIZE=6160),
           DSNAME=SYS12345.RAABC.LIBRARN.LIBRNIN,VOL=SER=(NONE)

NOTE: THE PROCEDURE SOURCE USED 0.29 SECONDS AND 412K.
NOTE: SAS USED 412K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Generating Control Cards for OS IEBCOPY: Example 8

The job below first produces control statements for the IBM utility program, IEBCOPY. Then IEBCOPY executes, copying selected members.

```
//jobname  JOB account,user
//        EXEC SAS
//EXALIB    DD DISP=SHR,DSN=SYS1.EXALIB
//CNTRL     DD DISP=(,PASS),
// DSN=&&SYSIN,UNIT=DISK,SPACE=(TRK,(3,1))
PROC SOURCE INDD=EXALIB OUTDD=CNTRL NODATA NOPRINT;
   SELECT  ISP: ;  EXCLUDE ISPTCM ISPBAD ISPJ-ISPU: ;
   SELECT  ADF: GARBAGE:-G: ;
   SELECT  IEF:;   EXCLUDE IEFU-IEFV:;
   SELECT  IDC: IKJ: ; EXCLUDE IKJEB: IKJEG: ;
   SELECT  NOTTHERE  MISS-MISSING: ;
   EXCLUDE BADNAME  BADTOO: ;
   FIRST  ' COPY INDD=((EXALIB,R)),OUTDD=NEWEXA';
   BEFORE '  SELECT MEMBER=XXXXXXXX -----------' 17;
   BEFORE '       S      M=XXXXXXXX ***ALIAS***' 17 ALIAS;
//COPY    EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=A
//SYSUT3    DD UNIT=DISK,SPACE=(CYL,(1,1))
//SYSUT4    DD UNIT=DISK,SPACE=(CYL,(1,1))
//EXALIB    DD DISP=SHR,DSN=SYS1.EXALIB
//NEWEXA    DD DISP=(,KEEP),DSN=NEW.EXALIB,UNIT=DISK,VOL=SER=ABC111,
//             SPACE=(CYL,(5,1,300)),DCB=SYS1.LINKLIB
//SYSIN     DD DISP=(OLD,DELETE),DSN=&&SYSIN
```

Output 47.10 Producing Control Statements for the IEBCOPY Utility

```
        S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM8  STEP SASBASE  PROC

NOTE: THE JOB SRCEXM8 HAS BEEN RUN UNDER RELEASE 5.XX OF SAS AT SAS INSTITUTE INC. (BASE) (00000000).

NOTE: SAS OPTIONS SPECIFIED ARE:
       SORT=4

NOTE: THE INITIALIZATION PHASE USED 0.08 SECONDS.
1             PROC SOURCE INDD=EXALIB OUTDD=CNTRL NODATA NOPRINT;
2                SELECT  ISP: ;  EXCLUDE ISPTCM ISPBAD ISPJ-ISPU: ;
3                SELECT  ADF: GARBAGE:-G: ;
4                SELECT  IEF:;   EXCLUDE IEFU-IEFV:;
5                SELECT  IDC: IKJ: ; EXCLUDE IKJEB: IKJEG: ;
6                SELECT  NOTTHERE  MISS-MISSING: ;
7                EXCLUDE BADNAME  BADTOO: ;
8                FIRST  ' COPY INDD=((EXALIB,R)),OUTDD=NEWEXA';
9                BEFORE '  SELECT MEMBER=XXXXXXXX -----------' 17;
10               BEFORE '       S      M=XXXXXXXX ***ALIAS***' 17 ALIAS;


NOTE: THE NODATA OPTION HAS BEEN SPECIFIED WITH OUTDD=;
      ONLY 'FIRST', 'BEFORE', 'AFTER', AND 'LAST'
      CONTROL CARDS WILL BE WRITTEN TO OUTDD=.

NOTE: NO MEMBER FOUND FOR ALIAS (IGG08116);
      IT WILL BE TREATED AS A NON-ALIAS MEMBER.

NOTE: THE FOLLOWING SELECTED MEMBER NAMES OR RANGES WERE NOT FOUND:

     ISP:
     GARBAGE:-G:
     NOTTHERE
     MISS-MISSING:

NOTE: THE FOLLOWING EXCLUDED MEMBER NAMES OR RANGES WERE NOT FOUND:

     ISPTCM
     ISPBAD
     ISPJ-ISPU:
```

(continued on next page)

```
(continued from previous page)

     BADNAME
     BADTOO:

          SOURCE LIBRARY DATA SET: INDD=EXALIB,BLKSIZE=19069,
          DSNAME=SYS1.EXALIB,VOL=SER=XYZRS2


    1316   MEMBERS DEFINED IN SOURCE LIBRARY.

     591   ALIASES DEFINED IN SOURCE LIBRARY.

     359   DIRECTORY BLOCKS ALLOCATED.

     346   DIRECTORY BLOCKS USED.

      67   MEMBERS SELECTED.

       0   RECORDS READ FROM SOURCE LIBRARY.

     168   RECORDS WRITTEN TO OUTDD=CNTRL,
          LABEL=(,SL),DISP=NEW,DCB=(BLKSIZE=6160),
```

```
2       S A S   L O G    OS SAS 5.XX          VS2/MVS JOB SRCEXM8  STEP SASBASE  PROC

          DSNAME=SYSABC.RADEF.SRCEXM8.SYSIN,VOL=SER=(NONE)

NOTE: THE PROCEDURE SOURCE USED 0.75 SECONDS AND 552K.
NOTE: SAS USED 552K MEMORY.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      PO BOX 8000
      CARY, N.C. 27511-8000
```

Output 47.11 IEBCOPY Output: Selected Members Copied

```
                                   IEBCOPY MESSAGES AND CONTROL STATEMENTS                    PAGE 0001


                    COPY INDD=((EXALIB,R)),OUTDD=NEWEXA
                     SELECT MEMBER=ADFIDF00 -----------
                     SELECT MEMBER=ADFMDFLT -----------
                     SELECT MEMBER=ADFMDF01 -----------
                     SELECT MEMBER=ADFMDF03 -----------
                     SELECT MEMBER=AKJLKL01 -----------
                          S      M=IKJLKL01 ***ALIAS***
                     SELECT MEMBER=IDCAM01  -----------
                          S      M=ALTER    ***ALIAS***
                          S      M=DEF      ***ALIAS***
                          S      M=DEFINE   ***ALIAS***
                          S      M=DEL      ***ALIAS***
                          S      M=DELETE   ***ALIAS***
                          S      M=EXP      ***ALIAS***
                          S      M=EXPORT   ***ALIAS***

 ... NOT ALL OUTPUT SHOWN ...

IEB167I  FOLLOWING MEMBER(S)  COPIED  FROM INPUT DATA SET REFERENCED BY EXALIB   -
IEB154I  ADFIDF00 HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMCOPY HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMDFLT HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMDF01 HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMDF03 HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMFIND HAS BEEN SUCCESSFULLY  COPIED
IEB154I  ADFMPUT  HAS BEEN SUCCESSFULLY  COPIED

 ... NOT ALL OUTPUT SHOWN ...

IEB154I  STATUS   HAS BEEN SUCCESSFULLY  COPIED
IEB154I  VERIFY   HAS BEEN SUCCESSFULLY  COPIED
IEB154I  VFY      HAS BEEN SUCCESSFULLY  COPIED
IEB154I  WHEN     HAS BEEN SUCCESSFULLY  COPIED
IEB154I  XPRA     HAS BEEN SUCCESSFULLY  COPIED
IEB144I  THERE ARE 0000027 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY NEWEXA
IEB149I  THERE ARE 0000268  UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB147I  END OF JOB -00 WAS HIGHEST SEVERITY CODE
```

REFERENCES

International Business Machines Corporation (1980), *OS/VS2 MVS Data Management Services Guide*, SRL Form Number GC26-3875.
International Business Machines Corporation (1977), *OS/VS2 MVS Utilities*, SRL Form Number GC26-3902.
International Business Machines Corporation (1979), *OS/VS2 MVS JCL*, SRL Form Number GC28-0692.
International Business Machines Corporation (1983), *VSE/Advanced Functions System Control Statements*, SRL Form Number SC33-6095.
International Business Machines Corporation (1979), *VSE/Advanced Functions System Management Guide*, SRL Form Number SC33-6094.

NOTES

1. **OS**: PROC SOURCE processes Partitioned Data Sets (PDSs) containing 80-byte records. Examples of such data sets include JCL procedure, macro, and statement libraries.

VSE: PROC SOURCE processes VSE source statement **sub**libraries. The term library, and the abbreviation SSL (Source Statement Library), in this chapter imply a sublibrary under VSE.

2. **OS**: SOURCE provides some general PDS directory processing facilities not limited to PDSs with 80-byte member records.

3. **OS**: SOURCE by default generates records for the IBM utility, IEBUPDTE, which reloads an unloaded data set. The following record is generated before each member in the unloaded data set:

```
./      ADD NAME=member
```

If the member has any aliases, SOURCE generates a record like this

```
./      ALIAS NAME=alias
```

for each alias after the member in the unloaded data set.

VSE: SOURCE by default generates records for the IBM utility program, MAINT, which reloads the unloaded library. PROC SOURCE generates the following records before each member in the unloaded data set:

```
CATALS A.membername
BKEND
```

and generates the following record after each member:

```
BKEND
```

You can construct input for other programs by using BEFORE, AFTER, FIRST, and LAST statements.

4. **OS**: the unloaded data set is suitable for reloading by IEBUPDTE or other source library maintenance utilities, including the ability to recognize and properly handle aliases.

VSE: the unloaded library is suitable for reloading by the MAINT library maintenance utility.

5. **VSE**: the file specified by the DIRDD= option will contain the entire SSL, not just a sublibrary.

6. **OS**: the list of member names includes any aliases.

7. **OS**: the OUTBLK= value is used only if neither the OUTDD= JCL DD statement nor the data set label/DSCB specifies a blocksize value.

8. **VSE**: you cannot use the colon (:) or the hyphen (-) notation with the SEARCH option.

9. **VSE**: you cannot use the colon (:) or the hyphen (-) notation with the SEARCH option.

10. See note 4.

11. **OS**: the beginning column can be any from 1 to 73. The field is 8 bytes long and contains aliases, as well as main member names.

12. **VSE**: right-aligns the member name in the 8-column field and pads it with blanks on the left. If the FULL option is specified, the sublibrary name and member name are right-aligned in a 10-column field and padded with blanks.

Chapter 48

The SUMMARY Procedure

Operating systems: All

ABSTRACT

The SUMMARY procedure computes descriptive statistics on numeric variables in a SAS data set and outputs the results to a new SAS data set.

INTRODUCTION

The SUMMARY procedure creates a SAS data set containing summary statistics. Each observation in the new data set contains the statistics for a different subgroup of the observations in the input data set. These subgroups represent all possible combinations of the levels of the variables in the CLASS statement.

Similarly, PROC MEANS also computes descriptive statistics with these important differences. MEANS produces subgroup statistics only when a BY statement

is used, and the input data must be sorted by the BY variables. For more than one grouping variable, several executions of MEANS are needed to produce the information that SUMMARY outputs in one execution.

SUMMARY does not produce any printed output. The SUMMARY output data set is typically printed with PROC PRINT or is input to a DATA step that extracts the desired information.

SPECIFICATIONS

The SUMMARY procedure is controlled by these statements:

PROC SUMMARY *options;*
CLASS *variables;*
VAR *variables;*
BY *variables;*
FREQ *variable;*
WEIGHT *variable;*
ID *variables;*
OUTPUT OUT=*SASdataset statistics;*

A VAR statement and an OUTPUT statement must be specified. The VAR statement must precede the OUTPUT statement. Only one OUTPUT statement is permitted.

PROC SUMMARY Statement

PROC SUMMARY *options;*

The options below can appear in the PROC SUMMARY statement:

DATA=*SASdataset*
: names the data set to be used by SUMMARY. If DATA= is omitted, the most recently created SAS data set is used.

MISSING
: requests that SUMMARY treat missing values as valid subgroup values for the CLASS variables.

NWAY
: specifies that statistics for only the observation with the highest _TYPE_ value (highest level of interaction among CLASS variables) are to be output.

IDMIN
: specifies that the value of the ID variable should be its minimum rather than its maximum value for the corresponding observations of the input data set.

DESCENDING
: specifies that the output data set is to be ordered by descending _TYPE_ value (ascending is the default). This causes the overall totals (_TYPE_=0) to be placed at the end of each BY group. This option has no effect if the NWAY option is also specified.

ORDER=FREQ
ORDER=DATA
ORDER=INTERNAL
ORDER=EXTERNAL | FORMATTED
: specifies the order in which you want the values of the CLASS variables to be sorted for the output data set. If you specify ORDER=FREQ, the values are ordered by descending frequency count so that class values occurring in the most observations come first. If you specify ORDER=DATA, class values are kept in the order encountered in the input data set. If you specify ORDER=INTERNAL, the values are ordered by their internal representation. If you specify ORDER=EXTERNAL or FORMATTED, the class values are ordered by their formatted (external representation). ORDER=INTERNAL is the default.

VARDEF=DF
VARDEF=WEIGHT | WGT
VARDEF=N
VARDEF=WDF
: specifies the divisor to be used in the calculation of the variance. VARDEF=DF requests that the degrees of freedom (N-1) be used as the divisor. VARDEF=WEIGHT or WGT requests that the sum of the weights be used. VARDEF=N requests that the number of observations (N) be used. VARDEF=WDF requests that the sum of the weights minus one be used. The default is VARDEF=DF.

CLASS Statement

CLASS *variables;*

In the CLASS or CLASSES statement you can name the variables to be used to form subgroups. The CLASS statement must be present to produce subgroup statistics. Without a CLASS statement SUMMARY produces only one observation per BY group.

VAR Statement

VAR *variables;*

In the VAR statement, give the numeric variables for which you want summary statistics. The VAR statement must precede the OUTPUT statement.

BY Statement

BY *variables;*

A BY statement can be used with PROC SUMMARY to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

The BY statement is not often used in PROC SUMMARY since the CLASS statement serves a similar but more general role without the need to sort. BY group processing requires more computer resources than CLASS variable processing.

FREQ Statement

FREQ *variable;*

The FREQ statement specifies a numeric variable in the input SAS data set. If a FREQ statement is used, each observation in the input data set is assumed to represent *n* observations, where *n* is the value of the FREQ variable. If the value is not an integer, it is truncated to its integer portion. If the FREQ value is less than one or is missing, the observation is skipped.

WEIGHT Statement

WEIGHT *variable;*

The WEIGHT statement specifies a numeric variable in the input SAS data set whose values are to be used to weight each observation. Only one variable can be specified. Both the FREQ and WEIGHT statements can be used. The WEIGHT variable values can be nonintegers and are used to calculate a weighted mean and a weighted variance. If the value of the WEIGHT variable is less than zero or is missing, a value of zero is assumed.

ID Statement

ID *variables;*

The ID statement includes additional variables in the output data set. If your ID statement names only one variable, the value of the ID variable for a given observation in the output data set is the maximum value it has in the corresponding observations of the input data set unless IDMIN is specified in the PROC statement. When your ID statement includes two or more variables, the maximum value is chosen as if the values of the ID variables were concatenated into one value for each observation. Thus the maximum value comes from only one of the corresponding observations in the input data set.

OUTPUT Statement

OUTPUT OUT=*SASdataset statistics;*

The options below can appear in the OUTPUT statement:

OUT=*SASdataset*
: names the output SAS data set. If OUT= is not specified, SUMMARY names the new data set as if OUT=_DATA_ were specified. If you want to create a permanent SAS data set, you must specify a two-level name. (See "SAS Files" for more information on permanent SAS data sets).

statistics
: specifies the statistics you want in the new data set and names the variables containing these statistics. Output requests have four forms:

keyword=names
keyword(variables)=names
keyword=
keyword(variables)=

Below is a list of the keywords and corresponding statistics that can be specified in the output requests:

N	number of observations in the subgroup with nonmissing values
NMISS	number of observations in the subgroup having missing values for the variable
MEAN	mean
STD	standard deviation
MIN	minimum value
MAX	maximum value
RANGE	range
SUM	sum
VAR	variance
USS	uncorrected sum of squares
CSS	corrected sum of squares
CV	coefficient of variation
STDERR	standard error of the mean
T	Student's *t* value for testing the hypothesis that the population mean is 0
PRT	probability of a greater absolute value for the Student's *t* value above
SUMWGT	sum of the WEIGHT variable values.

Formulas for these statistics are presented in "SAS Descriptive Procedures." To request statistics, give any number of keywords in the OUTPUT statement. The form of the output request determines the name of the new variable containing the statistic. Four choices are described below.

Output Form 1

keyword=names;

If you want the statistic's values to have names different from the names of the original variables, follow the equal sign with a list of new variable names. These names will then be used for the statistic's values in the new data set.

The first variable name following the equal sign is given to the corresponding statistic for the first variable in the VAR statement; the second name is given to the statistic for the second variable in the VAR statement; and so on, as in PROC MEANS.

For example, suppose that you want to calculate means for the variables PRE and POST. In the output data set named RESULTS, the variable containing the PRE mean is PREMEAN; the variable containing the POST mean is POSTMEAN.

```
PROC SUMMARY;
   CLASS TEACHER;
   VAR PRE POST;
   OUTPUT OUT=RESULTS MEAN=PREMEAN POSTMEAN;
```

Output Form 2

keyword(variables)=names;

You can calculate a statistic for only certain variables in the VAR statement and give those statistics new names in the output data set. After the keyword name, list the variables for which you want statistics in parentheses. Then place the names you assign to those statistics after the equal sign.

For example, suppose you want to compute standard deviations for only the POST variable, but want means for both variables. The following statements could be used to do this:

```
PROC SUMMARY;
   CLASS TEACHER;
   VAR PRE POST;
   OUTPUT OUT=RESULTS MEAN=PREMEAN POSTMEAN
   STD(POST)=STDPOST;
```

Output Form 3

keyword=;

The statistic in the new data set will have the same name as the corresponding variable in the input data set if you simply follow the keyword with an equal sign.

For example, say you want the output data set to contain means for the variables PRE and POST, and the variables containing the means are also to be called PRE and POST. You can specify the following:

```
PROC SUMMARY;
   CLASS TEACHER;
   VAR PRE POST;
   OUTPUT OUT=RESULTS MEAN=;
```

Do not use this form or the next one for more than one keyword in the OUTPUT statement. Multiple statistic requests using this form or the next one would cause the output data set to contain two or more variables with the same name. PROC SUMMARY cannot create duplicate variable names.

Output Form 4

keyword(variables)=;

You can compute the statistic for a subset of the variables in the VAR statement. After the keyword name, list the variables for which you want the statistic in parentheses. When no new variable names follow the equal sign, the statistics will have the same names as the original variables. Do not use this method for more than one statistic.

For example, say that you want the output data set to contain the mean for the variable PRE but standard deviations for both PRE and POST. The mean will have the name PRE, but the standard deviations will have new names.

```
PROC SUMMARY;
   VAR PRE POST;
   OUTPUT OUT=RESULTS MEAN(PRE)=  STD=STDPRE STDPOST;
```

DETAILS

Missing Values

If any CLASS variable has a missing value for an observation, that observation is not used in SUMMARY unless the MISSING option is specified in the PROC SUMMARY statement.

Limitations

The maximum number of combinations of CLASS levels is 32767 or 2^{15}-1. Because of computer resource limits the maximum number of CLASS variables is 24.

Output Data Set

SUMMARY produces an output data set and no printed output. The output data set's contents are described below.

Variables Variables in the output data set are

- the variables in the BY statement if a BY statement is used.
- the ID variables, if an ID statement is used.
- the variables in the CLASS statement. When formats combine several internal values into one formatted value, the lowest internal value is output.
- a variable created by SUMMARY named _TYPE_ whose values contain information about the CLASS variables that define each subgroup.
- a variable created by SUMMARY named _FREQ_ that gives the number of observations (both missing and nonmissing) for the current subgroup.
- the variables containing the subgroup statistics requested by the keywords in the OUTPUT statement.

For example, consider the data set created by SUMMARY with these statements:

```
PROC SUMMARY;
   CLASS TMT;
   VAR X Y;
   OUTPUT OUT=STATS  MEAN=MX MY  N=NX NY;
```

The data set STATS contains these variables:

TMT _TYPE_ _FREQ_ MX MY NX NY

Computer Resources

The SUMMARY procedure is designed to collect descriptive statistics in a very efficient manner, using memory resources as available.

SUMMARY requires approximately 250K for program space (this value is operating-system dependent) plus the maximum of the three following calculations divided by 1024. The result is the number of kilobytes required.

1. Sum of

   NLEVELS*(VLEN+FLEN+8)

   for each CLASS variable where
   - NLEVELS is the number of levels (values) for the CLASS variable
   - VLEN is the internal length for the CLASS variable
   - FLEN is the external (formatted) length for the CLASS variable.
   - Sum of

     NLEVELS*(VLEN+2)

     for each CLASS variable plus

     NINT*(10+(NC*4))

   where
   - NLEVELS and VLEN are defined above
   - NINT is the number of CLASS interactions found on the input data set

- NC is the number of CLASS variables.
- (2**NC)*(NC*2+28)+NINT*6

where

- NINT and NC are defined above.

Note that these calculations are approximate. If more memory is available, SUMMARY requires less workfile usage and is more efficient.

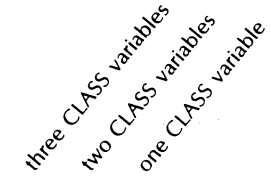
Observations

The number of observations in the output data set produced by PROC SUMMARY depends on the number of distinct values or levels (for example, a, b, c) of the CLASS variables (A, B, C).

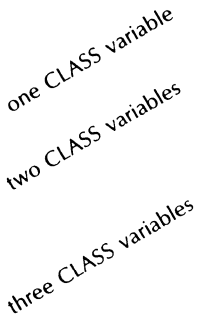
Observations in the data set produced by SUMMARY are identified by a variable named _TYPE_. Values of _TYPE_ indicate which subgroup (defined by values of the CLASS variables) produced the summary statistics in that observation in the output data set.

Table 48.1 shows how values of _TYPE_ are determined for one CLASS variable (A), two CLASS variables (A and B), and three CLASS variables (A, B, C). The same logic can be extended to analyses by PROC SUMMARY with more than three CLASS variables. The table assumes that all combinations of class levels occur in the data.

Table 48.1 Observations in PROC SUMMARY's Output Data Set



C	B	A	_TYPE_	Subgroup Defined By	Number of Observations of this _TYPE_ in the Data Set	Total Number of Observations in the Data Set
0	0	0	0	Total	1	
0	0	1	1	A	a	1+a
0	1	0	2	B	b	
0	1	1	3	A*B	a*b	1+a+b+a*b
1	0	0	4	C	c	
1	0	1	5	A*C	a*c	
1	1	0	6	B*C	b*c	1+a+b+a*b+c
1	1	1	7	A*B*C	a*b*c	+a*c+b*c+a*b*c
Binary equivalent of _TYPE_ value				A ,B ,C=CLASS variables	a, b, c=number of levels of A, B, C respectively	



For a CLASS statement with one class variable,

```
CLASS A;
```

the output data set contains one observation for totals (_TYPE_=0) and an observation for each level of A (_TYPE_=1).

For a CLASS statement with two class variables,

```
CLASS B A;
```

an observation for each level of B (_TYPE_=2) and an observation for each level of A*B (_TYPE_=3) are added.

For a CLASS statement with three class variables,

```
CLASS C B A;
```

observations for levels of C (_TYPE_=4), levels of A*C (_TYPE_=5), B*C (_TYPE_=6), and A*B*C (_TYPE_=7) are added.

You can extend the table to an N-way analysis by adding columns to the table on the left. A value of one is entered in a column if that CLASS variable is involved in defining the subgroup, a zero otherwise. Note that the ones and zeros in the CLASS variable columns form binary numbers across the row for each subgroup.

The decimal equivalent of one of these binary numbers for a subgroup is the _TYPE_ value for that subgroup. There are 2^n different values of _TYPE_ produced by SUMMARY for a given analysis, where n is the number of CLASS variables. The number of observations of each _TYPE_ in the data set depends on the number of levels of the CLASS variables as shown in the table above.

Consider the case where the CLASS statement contains two variables, SEX and RACE. The variable SEX has the two values F and M; the variable RACE has the two values B and W. The observations in the output data set are determined as in **Table 48.2**.

Table 48.2 Two CLASS Variables: PROC SUMMARY

Sex	Race	_TYPE_	Subgroup Defined By	Number of Observations of this _TYPE_ in the Data Set	Total Number of Observations in the Data Set
0	0	0	Total	1	
0	1	1	Race	2	
1	0	2	Sex	2	
1	1	2	Race*Sex	4	9

These statements

```
PROC SUMMARY;
   CLASS SEX RACE;
```

```
   VAR AGE;
   OUTPUT OUT=STATS MEAN=AGEMEAN;
```

produce the data set in **Table 48.2**.

The _TYPE_=0 observation contains the overall means for the data set. The SEX and RACE values for this output observation are missing. The value of _FREQ_ is 10 because ten observations were used to form this group.

The _TYPE_=1 observations contain the group means for the rightmost variable in the CLASS statement, in this case RACE. The first of these contains the mean for the four observations having a value of B for RACE; the second contains the mean for the six observations having the value of W for RACE.

The _TYPE_=2 observations contain the means for the two groups of observations defined by the SEX values.

The _TYPE_=3 observations contain the means for the subgroups defined by the combinations of the RACE and SEX values. For example, the first observation in this set contains the mean for the two observations having a SEX value of F and a RACE value of B.

Usage Note: SAS Bit-Testing Feature

The _TYPE_ variable is a good candidate for use with the SAS bit-testing feature. For example, say that you want to print a report using just the observations with the B and C combinations:

```
PROC SUMMARY
   CLASS A B C;
   VAR X Y Z;
   OUTPUT OUT=STATS MEAN= ;
DATA _NULL_;
   SET STATS;
   IF _TYPE_='011'B;
   .
   .
more SAS statements
```

The IF statement could also be written:

```
IF _TYPE_=3;
```

EXAMPLES

Sales Data: Example 1

Below are the statements that ask SUMMARY to create an output data set containing summary statistics for two CLASS variables, PRODUCT and DATE. PROC PRINT prints the summary file.

```
PROC FORMAT;
   VALUE MMM 1='JAN' 2='FEB' 3='MAR' 4='APR' 5='MAY' 6='JUN'
             7='JUL' 8='AUG' 9='SEP' 10='OCT' 11='NOV' 12='DEC';
   more SAS statements
;
DATA SALES2;
   SET SALES;
   KEEP PRODUCT DATE YEAR UNITS;
PROC SUMMARY DATA=SALES;
   CLASS PRODUCT DATE;
```

```
   VAR UNITS;
   ID YEAR;
   OUTPUT OUT=B SUM=SALES;
PROC PRINT DATA=B;
```

Output 48.1 Output Data Set Containing Summary Statistics

1

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
1	1981		.	0	742	4781
2	1979		01JAN79	1	8	67
3	1979		01FEB79	1	10	69
4	1979		01MAR79	1	12	82
5	1979		01APR79	1	13	77
6	1979		01MAY79	1	10	67
7	1979		01JUN79	1	12	55
8	1979		01JUL79	1	22	108
9	1979		01AUG79	1	21	129
10	1979		01SEP79	1	22	125
11	1979		01OCT79	1	21	145
12	1979		01NOV79	1	19	90
13	1979		01DEC79	1	14	75
14	1980		01JAN80	1	26	144
15	1980		01FEB80	1	22	141
16	1980		01MAR80	1	22	161
17	1980		01APR80	1	26	173
18	1980		01MAY80	1	20	135
19	1980		01JUN80	1	20	105
20	1980		01JUL80	1	23	130
21	1980		01AUG80	1	21	96
22	1980		01SEP80	1	22	134
23	1980		01OCT80	1	23	126
24	1980		01NOV80	1	16	93
25	1980		01DEC80	1	19	118
26	1981		01JAN81	1	19	143
27	1981		01FEB81	1	21	135
28	1981		01MAR81	1	33	200
29	1981		01APR81	1	34	181
30	1981		01MAY81	1	27	181
31	1981		01JUN81	1	36	220
32	1981		01JUL81	1	30	254
33	1981		01AUG81	1	19	202
34	1981		01SEP81	1	26	206
35	1981		01OCT81	1	24	187
36	1981		01NOV81	1	14	113
37	1981		01DEC81	1	15	114
38	1981	4005	.	2	212	1967
39	1981	8401	.	2	270	2062
40	1981	9054	.	2	260	752
41	1979	4005	01JAN79	3	5	56
42	1979	4005	01FEB79	3	5	57
43	1979	4005	01MAR79	3	5	60
44	1979	4005	01APR79	3	5	51
45	1979	4005	01MAY79	3	5	49
46	1979	4005	01JUN79	3	5	35
47	1979	4005	01JUL79	3	5	36
48	1979	4005	01AUG79	3	5	42
49	1979	4005	01SEP79	3	5	34
50	1979	4005	01OCT79	3	5	48
51	1979	4005	01NOV79	3	5	41
52	1979	4005	01DEC79	3	5	29
53	1980	4005	01JAN80	3	5	37
54	1980	4005	01FEB80	3	6	49

2

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
55	1980	4005	01MAR80	3	6	63
56	1980	4005	01APR80	3	6	59
57	1980	4005	01MAY80	3	6	43
58	1980	4005	01JUN80	3	6	42
59	1980	4005	01JUL80	3	6	42
60	1980	4005	01AUG80	3	6	32
61	1980	4005	01SEP80	3	6	48

(continued on next page)

(continued from previous page)

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
62	1980	4005	01OCT80	3	6	58
63	1980	4005	01NOV80	3	6	56
64	1980	4005	01DEC80	3	6	42
65	1981	4005	01JAN81	3	6	67
66	1981	4005	01FEB81	3	7	57
67	1981	4005	01MAR81	3	7	86
68	1981	4005	01APR81	3	7	70
69	1981	4005	01MAY81	3	7	81
70	1981	4005	01JUN81	3	7	56
71	1981	4005	01JUL81	3	7	84
72	1981	4005	01AUG81	3	7	87
73	1981	4005	01SEP81	3	7	82
74	1981	4005	01OCT81	3	7	81
75	1981	4005	01NOV81	3	6	64
76	1981	4005	01DEC81	3	6	43
77	1979	8401	01JUL79	3	8	44
78	1979	8401	01AUG79	3	9	65
79	1979	8401	01SEP79	3	8	68
80	1979	8401	01OCT79	3	10	75
81	1979	8401	01NOV79	3	7	27
82	1979	8401	01DEC79	3	5	35
83	1980	8401	01JAN80	3	7	60
84	1980	8401	01FEB80	3	5	39
85	1980	8401	01MAR80	3	7	60
86	1980	8401	01APR80	3	10	68
87	1980	8401	01MAY80	3	8	63
88	1980	8401	01JUN80	3	7	50
89	1980	8401	01JUL80	3	8	60
90	1980	8401	01AUG80	3	6	46
91	1980	8401	01SEP80	3	9	72
92	1980	8401	01OCT80	3	8	49
93	1980	8401	01NOV80	3	6	29
94	1980	8401	01DEC80	3	6	55
95	1981	8401	01JAN81	3	7	63
96	1981	8401	01FEB81	3	7	60
97	1981	8401	01MAR81	3	14	87
98	1981	8401	01APR81	3	17	91
99	1981	8401	01MAY81	3	14	87
100	1981	8401	01JUN81	3	18	134
101	1981	8401	01JUL81	3	13	151
102	1981	8401	01AUG81	3	10	110
103	1981	8401	01SEP81	3	12	109
104	1981	8401	01OCT81	3	11	96
105	1981	8401	01NOV81	3	6	44
106	1981	8401	01DEC81	3	7	65
107	1979	9054	01JAN79	3	3	11
108	1979	9054	01FEB79	3	5	12

OBS	YEAR	PRODUCT	DATE	_TYPE_	_FREQ_	SALES
109	1979	9054	01MAR79	3	7	22
110	1979	9054	01APR79	3	8	26
111	1979	9054	01MAY79	3	5	18
112	1979	9054	01JUN79	3	7	20
113	1979	9054	01JUL79	3	9	28
114	1979	9054	01AUG79	3	7	22
115	1979	9054	01SEP79	3	9	23
116	1979	9054	01OCT79	3	6	22
117	1979	9054	01NOV79	3	7	22
118	1979	9054	01DEC79	3	4	11
119	1980	9054	01JAN80	3	14	47
120	1980	9054	01FEB80	3	11	53
121	1980	9054	01MAR80	3	9	38
122	1980	9054	01APR80	3	10	46
123	1980	9054	01MAY80	3	6	29
124	1980	9054	01JUN80	3	7	13
125	1980	9054	01JUL80	3	9	28
126	1980	9054	01AUG80	3	9	18
127	1980	9054	01SEP80	3	7	14
128	1980	9054	01OCT80	3	9	19
129	1980	9054	01NOV80	3	4	8
130	1980	9054	01DEC80	3	7	21
131	1981	9054	01JAN81	3	6	13
132	1981	9054	01FEB81	3	7	18
133	1981	9054	01MAR81	3	12	27
134	1981	9054	01APR81	3	10	20
135	1981	9054	01MAY81	3	6	13
136	1981	9054	01JUN81	3	11	30
137	1981	9054	01JUL81	3	10	19
138	1981	9054	01AUG81	3	2	5

(continued on next page)

(continued from previous page)

```
 139    1981    9054    01SEP81    3        7       15
 140    1981    9054    01OCT81    3        6       10
 141    1981    9054    01NOV81    3        2        5
 142    1981    9054    01DEC81    3        2        6
```

Census Data: Example 2

Data for the following example are selected characteristics of U.S. cities with populations over 50,000 according to the County and City Data Book, 1983. The class variables are POPGRP with three levels and REGION with four levels. The OUTPUT statement specifies statistics for the dependent variables with several different kinds of output requests.

Another example in "PROC Step Applications" shows the DATA step that read this census data into a SAS data set. PROC UNIVARIATE is used to determine cutoff points for the POPGRP variable. The TABULATE procedure is used to demonstrate another approach to these same data.

```
DATA CITIES;     (see "PROC Step Applications"
                                  for complete DATA step)
more SAS statements
;
PROC SUMMARY DATA=CITIES;
   CLASS POPGRP REGION;
   VAR OVER65 INCOME HOUSING ELECTRIC CRIME;
   OUTPUT OUT=CITYDATA N=NOVER65 NINCOME NHOUS NELEC NCRIME MEAN=
      STD(INCOME HOUSING ELECTRIC)=INCOMSTD HOUSESTD ELECSTD;
   FORMAT POPGRP PC.;
PROC PRINT DATA=CITYDATA;
   TITLE 'CENSUS DATA: RESULTS FROM SUMMARY PROCEDURE';
```

Output 48.2 Census Data: PROC SUMMARY

CENSUS DATA: RESULTS FROM SUMMARY PROCEDURE 37

OBS	POPGRP	REGION	_TYPE_	_FREQ_	NOVER65	NINCOME	NHOUS	NELEC	NCRIME	OVER65	INCOME	HOUSING	ELECTRIC	CRIME	INCOMSTD	HOUSESTD	ELECSTD
1	.		0	418	418	418	418	418	418	20552.9	7569.34	71406	47.8885	7856.02	1627.25	177443	11.7452
2	.	NORTH CENTRAL	1	112	112	112	112	112	112	18935.4	7853.63	64422	43.3684	7525.53	1551.28	124931	7.0366
3	.	NORTHEAST	1	74	74	74	74	74	74	30571.3	6709.34	91026	58.1432	7129.92	1285.77	346647	11.5923
4	.	SOUTH	1	114	114	114	114	114	114	19860.9	7098.77	73108	43.5652	8536.21	1250.58	88274	7.7512
5	.	WEST	1	118	118	118	118	118	118	16474.1	8293.46	64085	49.9245	7967.92	1822.84	120687	13.9414
6	50,001-75,000		2	162	162	162	162	162	162	7044.2	7778.08	23675	47.9718	6737.57	1890.68	4024	11.8870
7	75,001-100,000		2	87	87	87	87	87	87	9843.2	7483.18	32799	49.6853	7667.21	1705.08	5872	9.8244
8	OVER 100,000		2	169	169	169	169	169	169	39015.5	7413.60	137034	46.8836	9025.34	1260.27	266117	12.4458
9	50,001-75,000	NORTH CENTRAL	3	46	46	46	46	46	46	6482.2	8255.61	22567	43.8065	5989.65	1863.10	3403	6.5461
10	50,001-75,000	NORTHEAST	3	33	33	33	33	33	33	8785.5	6903.67	24131	57.6345	5841.85	1044.48	3513	12.1465
11	50,001-75,000	SOUTH	3	40	40	40	40	40	40	7386.1	7144.55	24738	42.4635	8091.20	1263.73	3712	8.7433
12	50,001-75,000	WEST	3	43	43	43	43	43	43	5990.9	8527.63	23524	50.1360	6965.88	2421.11	4986	13.6489
13	75,001-100,000	NORTH CENTRAL	3	27	27	27	27	27	27	9605.7	7889.59	32066	43.1026	8012.70	1520.11	4248	7.0382
14	75,001-100,000	NORTHEAST	3	18	18	18	18	18	18	12179.1	6652.89	34101	56.4678	6976.72	1743.41	3246	7.9960
15	75,001-100,000	SOUTH	3	13	13	13	13	13	13	11785.0	6621.85	34460	44.7631	8311.46	1548.09	10536	6.0896
16	75,001-100,000	WEST	3	29	29	29	29	29	29	7743.8	8006.28	31927	53.8107	7485.31	1637.41	5611	9.7728
17	OVER 100,000	NORTH CENTRAL	3	39	39	39	39	39	39	40082.7	7354.59	136191	43.0356	8999.79	941.13	193427	7.7200
18	OVER 100,000	NORTHEAST	3	23	23	23	23	23	23	76223.2	6474.70	231557	60.1843	9097.91	1197.22	607198	13.2233
19	OVER 100,000	SOUTH	3	61	61	61	61	61	61	29762.1	7170.39	113063	44.0323	8875.92	1171.39	105536	7.3962
20	OVER 100,000	WEST	3	46	46	46	46	46	46	31777.5	8255.61	122274	47.2767	9208.85	1180.18	179241	15.9719

REFERENCES

County and City Data Book, 1983 Files on Tape (machine-readable data file)/ prepared by the Bureau of the Census. Washington: The Bureau (producer and distributor), 1984.

County and City Data Book, 1983 Files on Tape Technical Documentation/ prepared by the Data User Services Division, Bureau of the Census. Washington: The Bureau, 1984.

Chapter 49

The TABULATE Procedure

Operating systems: All

ABSTRACT

PROC TABULATE constructs tables of descriptive statistics from compositions of classification variables, analysis variables, and statistics keywords. Tables can have up to three dimensions: column, row, and page.

INTRODUCTION

PROC TABULATE displays descriptive statistics in hierarchical tables. Each table cell belongs to a particular category of observations composed by crossing variable names. The statistic associated with each cell is calculated on values from all observations in that category. The statistics that PROC TABULATE computes are many of the same statistics computed by other descriptive procedures such as MEANS, FREQ, and SUMMARY. PROC TABULATE provides

- simple but powerful methods to create user-defined tables
- a great degree of flexibility in classification hierarchies
- a variety of mechanisms for titling and formatting variables and procedure-generated statistics.

Introductory Example

A Sample data set In the following examples, tables are presented that contain demographic information extracted from a data set containing the following variables:

Variable	Description
REGION	code for region of the country
CITYSIZE	code for relative population size (S=small, M=medium, L=large)
POP	urban population

Each observation contains data for one city.

In the examples below, table format is specified with the TABLE statement. (Most applications also use CLASS and VAR statements in addition to the PROC TABULATE statement.) The first example produces a single column with population figures for each region

```
PROC TABULATE;
   CLASS REGION;
   VAR POP;
   TABLE REGION, POP;
```

Output 49.1 Specifying TABLE REGION, POP

	POP
	SUM
REGION	
NC	4650000.00
NE	6666000.00
SO	6864000.00
WE	8376000.00

The next example produces a cross-tabulation of population values for REGION by CITYSIZE. These values can also be obtained by using PROC FREQ with a WEIGHT statement.

```
PROC TABULATE;
   CLASS REGION CITYSIZE;
   VAR POP;
   TABLE REGION, CITYSIZE*POP*SUM;
```

Output 49.2 Specifying TABLE REGION, CITYSIZE*POP*SUM

	CITYSIZE		
	L	M	S
	POP	POP	POP
	SUM	SUM	SUM
REGION			
NC	3750000.00	750000.00	150000.00
NE	5022000.00	1422000.00	222000.00
SO	4488000.00	2088000.00	288000.00
WE	5592000.00	2592000.00	192000.00

By changing the TABLE statement, the same values can be displayed in one column as shown below:

```
TABLE REGION*CITYSIZE, POP*SUM;
```

Output 49.3 Specifying TABLE REGION*CITYSIZE, POP*SUM;

		POP
		SUM
REGION	CITYSIZE	
NC	L	3750000.00
	M	750000.00
	S	150000.00
NE	CITYSIZE	
	L	5022000.00
	M	1422000.00
	S	222000.00
SO	CITYSIZE	
	L	4488000.00
	M	2088000.00
	S	288000.00
WE	CITYSIZE	
	L	5592000.00
	M	2592000.00
	S	192000.00

Requesting MEAN as well as SUM produces two columns.

```
TABLE REGION*CITYSIZE, POP*(SUM MEAN);
```

Output 49.4 Specifying TABLE REGION*CITYSIZE, POP*(SUM MEAN);

		POP	
		SUM	MEAN
REGION	CITYSIZE		
NC	L	3750000.00	625000.00
	M	750000.00	125000.00
	S	150000.00	25000.00
NE	CITYSIZE		
	L	5022000.00	837000.00
	M	1422000.00	237000.00
	S	222000.00	37000.00
SO	CITYSIZE		
	L	4488000.00	748000.00
	M	2088000.00	348000.00
	S	288000.00	48000.00
WE	CITYSIZE		
	L	5592000.00	932000.00
	M	2592000.00	432000.00
	S	192000.00	32000.00

Fundamentals

An understanding of the TABLE statement is fundamental to using TABULATE. The following are important elements of the TABLE statement:

- types of variables—classification and analysis
- TABLE statement operators (comma, blank space, asterisk, brackets, parentheses)
- table dimensions—pages, rows, columns.

Types of variables PROC TABULATE operates on two types of variables: classification (or class) variables and analysis variables. Class variables typically have a few discrete or noncontinuous values that define the classification. In the sample data set referred to above, REGION and CITYSIZE are class variables. Analysis variables have values on which statistics are calculated. In the sample data set, POP is used as an analysis variable.

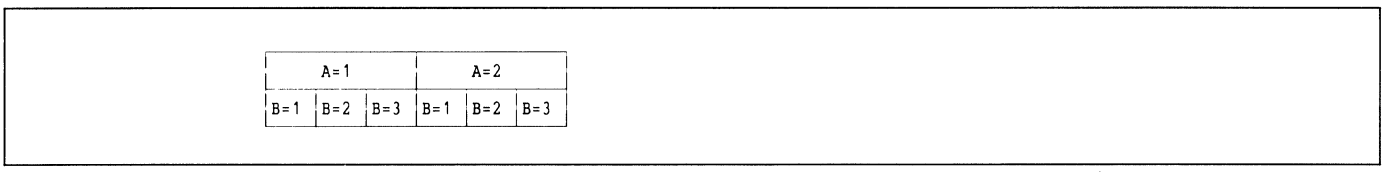
Table statement operators A table is an organized collection of cells. Four items define the contents of each cell:

- classification levels (from class variables)
- an analysis variable
- a statistic
- a format specification.

For example, a cell may contain the SUM (a statistic) of population values (an analysis variable) of small cities (CITYSIZE='S', a classification variable) in the South (REGION= 'SO', a classification variable). The cell value is formatted using the *w.d* specification 12.0. See **Specifications** for a discussion of the TABLE statement and a list of the statistic keywords. See **Details** for a discussion of format specifications.
You can define the cells of a table using the four items discussed above and the TABLE statement operators.

- Crossing (the * operator) compounds items into a hierarchy.
- Concatenation (the blank space operator) joins items one after another.
- Grouping (parentheses) controls the precedence of the other two operations.

The crossing operation arranges all levels of its second operand within each level of its first operand. For two class variables A and B, with A having two levels and B having three levels, "A crossed with B" is written **A*B** and produces the following:



Using the same data levels, B crossed with A is written **B*A**, and produces the following hierarchy:

B=1		B=2		B=3	
A=1	A=2	A=1	A=2	A=1	A=2

The concatenation operation causes all levels of its second operand to follow all levels of its first operand. For the same class variables and levels used above, B concatenated with A (written **A B**) produces:

A=1	A=2	B=1	B=2	B=3

You can also write compound expressions, which involve several operations. For example, **A*B C** (where the class variable C has three levels) produces:

A=1			A=2					
B=1	B=2	B=3	B=1	B=2	B=3	C=1	C=2	C=3

In compound expressions, the crossing operator (*) takes precedence over concatenation. You can use parentheses to alter the binding produced by this operator precedence. For example, the operations in the expression above are distributed differently if you write **A*(B C)**, which produces:

A=1						A=2					
B=1	B=2	B=3	C=1	C=2	C=3	B=1	B=2	B=3	C=1	C=2	C=3

Table dimensions TABULATE produces tables that can contain up to three dimensions:

- page (or wafer)
- row (or stub, side)
- column (or banner, top).

Each dimension is specified by an expression, and dimension expressions are separated by a comma. If all the dimensions are included, the first expression defines the page dimension, the second the row dimension, and the third the column dimension. You can also specify tables that contain only rows and columns or only columns.

Concatenation and crossing can be expressed in any of the three dimensions. For example, concatenation in the column dimension causes all values of the left name or crossing to be followed horizontally by all values of the name or crossing to its right. Thus, the concatenation **A B** produces:

A=1	A=2	B=1	B=2	B=3

Crossing in the column dimension is expressed vertically. Therefore, **A B*C** produces:

		B=1			B=2			B=3		
A=1	A=2	C=1	C=2	C=3	C=1	C=2	C=3	C=1	C=2	C=3

Concatenation in the row dimension causes all values of the left name or crossing to be followed vertically by all values of the name or crossing to its right. In the row dimension, crossing is expressed horizontally. For example:

A=1
A=2
B=1
B=2
B=3

A=1	
A=2	
B=1	C=1
	C=2
	C=3
B=2	C=1
	C=2
	C=3
B=3	C=1
	C=2
	C=3

In the page dimension, a crossing specifies the set of class values, statistics, and variable information for each logical page. When any classification value, analysis variable, statistic, or class variable in the page dimension changes, TABULATE begins a new logical page. Therefore, crossing is expressed verbally in the page titles and concatenation is expressed by beginning a new logical page. Pages for

the left name or crossing are followed by pages for the name or crossing to its right.

You can combine dimensions (for example, row * column) by crossing dimensions with each other, column within row and row within page, in the definition of individual table cell values. Each cell corresponds to one statistic calculated on all values of an analysis variable from observations sharing the same set of crossed class variable values. As this statement implies, you cannot cross an analysis variable with another analysis variable or a statistic with another statistic. These invalid crossings would result in two analysis variables or two statistics defining the same table cell. In the above examples, both the analysis variable and the statistic appear in the column dimension. The set of analysis variables and the set of statistics may appear together in any dimension. However, all statistics must appear in only one dimension, and all analysis variables must appear in only one dimension.

The following example shows the analysis variable in the column dimension and the statistics in the row dimension:

```
CLASS REGION CITYSIZE;
VAR POP;
TABLE REGION*CITYSIZE*(SUM MEAN), POP;
```

Output 49.5 Table with the Analysis Variable in the Column Dimension and the Statistics in the Row Dimension

1

			POP
REGION	CITYSIZE		
NC	L	SUM	3750000.00
		MEAN	625000.00
	M	SUM	750000.00
		MEAN	125000.00
	S	SUM	150000.00
		MEAN	25000.00
NE	CITYSIZE		
	L	SUM	5022000.00
		MEAN	837000.00
	M	SUM	1422000.00
		MEAN	237000.00
	S	SUM	222000.00
		MEAN	37000.00
SO	CITYSIZE		
	L	SUM	4488000.00
		MEAN	748000.00
	M	SUM	2088000.00
		MEAN	348000.00
	S	SUM	288000.00
		MEAN	48000.00
WE	CITYSIZE		
.			.
.			.
.			.

Next, add four additional variables to the example: PRODUCT and SALETYPE are class variables; QUANTITY and AMOUNT (price) are analysis variables.

The desired report contains a one-page table for each product. The table provides an analysis of wholesale and retail sales by REGION and by CITYSIZE.

```
TABLE PRODUCT, REGION CITYSIZE, SALETYPE*(QUANTITY AMOUNT);
```

Output 49.6 Producing a One-page Table for Each Product

1

PRODUCT A100

	SALETYPE			
	R		W	
	QUANTITY	AMOUNT	QUANTITY	AMOUNT
	SUM	SUM	SUM	SUM
REGION				
NC	1250.00	31250.00	1250.00	25000.00
NE	1600.00	40000.00	1600.00	32000.00
SO	1880.00	47000.00	1880.00	37600.00
WE	1840.00	46000.00	1840.00	36800.00
CITYSIZE				
L	3190.00	79750.00	3190.00	63800.00
M	2440.00	61000.00	2440.00	48800.00
S	940.00	23500.00	940.00	18800.00

2

PRODUCT A200

	SALETYPE			
	R		W	
	QUANTITY	AMOUNT	QUANTITY	AMOUNT
	SUM	SUM	SUM	SUM
REGION				
NC	1295.00	32375.00	1295.00	25900.00
NE	1645.00	41125.00	1645.00	32900.00
SO	1925.00	48925.00	1925.00	38500.00
WE	1885.00	47125.00	1885.00	37700.00
CITYSIZE				
L	3250.00	81250.00	3250.00	65000.00
M	2500.00	63500.00	2500.00	50000.00
S	1000.00	24800.00	1000.00	20000.00

When you specify an analysis variable but no statistic, the default statistic is SUM. If you specify neither an analysis variable nor a statistic, the default statistic is N, the frequency of occurrence of an interaction of crossed class values.

In the table in **Output 49.6**, the page dimension is PRODUCT; the row dimension is REGION CITYSIZE, a concatenation; the column dimension is SALETYPE *(QUANTITY AMOUNT). The pages for the first two products follow.

Important Features

The universal classifier ALL The universal class variable, ALL, represents a special class that has only one value. ALL is useful if you want to include statistics on totals and sub-totals within a classification group in a table. See **Details** for more information.

Percentages Percentages are calculated on two types of statistics. The two keywords are PCTN and PCTSUM. PCTN is calculated by dividing the frequency (N) for a subgroup of observations by the frequency of a containing group. PCTSUM is calculated by dividing the SUM of an analysis variable's values in a sub-group by the SUM of an analysis variable's values in a containing group. The two keywords PCTN and PCTSUM are used as if they were statistics keywords. It is not necessary to request N or SUM in order to request PCTN or PCTSUM. Each percentage is calculated on a denominator defined by crossings of classification and analysis variables. Denominator definitions follow the percent keyword in brackets (<>). See **Details** for more information. If percent keywords are labeled in the TABLE statement (see example below), the label specification follows the bracketed denominator definition list. For example, these statements produce **Output 49.7**.

```
TABLE (REGION*CITYSIZE ALL),
   (QUANTITY AMOUNT)*
      (SUM*F=8. PCTSUM<REGION*CITYSIZE ALL>='PERCENT'*F=8.2);
```

Output 49.7 Percents Calculated on Two Types of Statistics

		QUANTITY		AMOUNT	
		SUM	PERCENT	SUM	PERCENT
REGION	CITYSIZE				
NC	L	4544	11.38	102240	11.37
	M	2132	5.34	47920	5.33
	S	944	2.36	21240	2.36
NE	CITYSIZE				
	L	4842	12.13	108945	12.12
	M	3650	9.14	82125	9.14
	S	1246	3.12	28035	3.12
SO	CITYSIZE				
	L	4606	11.54	103635	11.53
	M	4298	10.77	97705	10.87
	S	2508	6.28	56230	6.25
WE	CITYSIZE				
	L	5310	13.30	119475	13.29
	M	4720	11.82	106200	11.81
	S	1122	2.81	25245	2.81
ALL		39922	100.00	898995	100.00

Titling and formatting There are four sources of text for page, row, and column titles:

- formatted class values
- literals attached to variable or keyword names in the TABLE statement
- variable names or labels (for both class and analysis variables)
- keyword names or labels (for statistics and ALL).

See **Details** for the use of each of these sources.

SPECIFICATIONS

The TABULATE procedure is controlled by the following statements:

PROC TABULATE *options;*
CLASS *variables;*
VAR *variables;*
FREQ *variable;*
WEIGHT *variable;*
FORMAT *variables format.;*
LABEL *variable=label;*
BY *variables;*
TABLE {*dimension_expression,*} {*dimension_expression,*}
dimension_expression/options;
KEYLABEL *keyword='text'...;*

The PROC TABULATE statement is always accompanied by one or more TABLE statements specifying the tables to be produced. Classification variables used in TABLE statements must be specified in the CLASS statement. Analysis variables used in TABLE statements must appear in the VAR statement. A variable that appears in both a CLASS statement and a VAR statement is used as a CLASS variable only. The WEIGHT, FREQ, and BY statements are optionally specified once for the entire procedure step.

PROC TABULATE Statement

PROC TABULATE *options;*

The options below can appear on the PROC TABULATE statement:

DATA=*SASdataset*
: specifies the SAS data set to be used by TABULATE. If DATA= is not specified, TABULATE uses the most recently created SAS data set.

MISSING
: requests that missing values be considered as valid levels for the classification variables. Special missing values are considered as different level values. Unless MISSING is specified, TABULATE does not include observations with a missing value for one or more classification variables in the analysis.

FORMAT=*formatname*
: specifies a default format for the formatting of each table cell. You can use any valid SAS or user-defined format. If no value is specified, the default value is 12.2. The default format is always overridden by any formats specified in a TABLE statement. This option is especially useful for decreasing the number of print positions required to print a table.

ORDER=FREQ
ORDER=DATA
ORDER=INTERNAL
ORDER=FORMATTED
: specifies the order in which the class variable values are displayed in each table. If ORDER=FREQ, the class values are ordered by descending frequency count so that class values occurring in the greatest number of observations come first. If ORDER=DATA, class values are kept in the order they were encountered in the data set. If ORDER=INTERNAL, the class values are ordered by the internal representation of the values. If ORDER=FORMATTED, the class values are ordered by the formatted (external representation) of the value. If you omit ORDER= or give an unrecognized value, TABULATE orders by the internal value.

FORMCHAR{*index list*}='*string*'
: defines the characters to be used for constructing the table outlines and dividers. The value is a string 11 characters long defining the two bar characters, vertical and horizontal, and the nine corner characters: upper left, upper middle, upper right, middle left, middle middle (cross), middle right, lower left, lower middle, and lower right. The default value is FORMCHAR='|----|+|---'. Any character or hexadecimal string can be substituted to customize the table appearance. Use an index list to specify which default form character each supplied character replaces; or replace the entire default string by specifying the full 11 character replacement string with no index list. For example, change the four corners to asterisks by using

```
FORMCHAR(3 5 9 11)= '****'   .
```

Specifying

```
FORMCHAR='           ' (11 blanks)
```

produces tables with no outlines or dividers. If you have your printout routed to an IBM 6670 printer using an extended font (typestyle 27 or 225) with input character set 216, we recommend:

```
FORMCHAR='FABFACCCBCEB8FECABCBBB'X   .
```

If you are printing on a printer with a TN (text) print train, we recommend:

```
FORMCHAR='4FBFACBFBC4F8F4FABBFBB'X   .
```

See the CALENDAR procedure for an illustration of these characters.

DEPTH=*number*
: specifies the maximum depth of any crossing in a dimension. The default depth is 10. The depth of a crossing refers to the number of elements that are crossed with each other. For example, the depth of A*B is two and the maximum depth of the dimension A*(B*N ALL) is three since A*B*N has depth three.

CLASS Statement

CLASS *variables*;

The CLASS statement is used to identify variables in the input data set as classification variables. Any classification variable used in a TABLE statement must have

been previously listed in the CLASS statement. The variables may have either numeric or character values. Normally each class variable has a small number of discrete values or unique levels. Continuous values for a numeric variable can be grouped into discrete levels by using PROC FORMAT.

VAR Statement

VAR *variables;*

The VAR statement is used to identify analysis variables in the input data set. Analysis variables must be numeric and may contain continuous values. All analysis variables used in TABLE statements must be included in this list.

FREQ and WEIGHT Statements

FREQ *variable;*
WEIGHT *variable;*

Both the FREQ and WEIGHT statements specify a numeric variable in the input SAS data set whose values are used to weight each observation. Only one variable can be used in each statement; both statements can be used.

If the FREQ statement is used, each observation in the input data set is assumed to represent *n* observations, where *n* is the value of the FREQ variable. If the value is not an integer, the value is truncated to the integer portion. If the FREQ variable has a value less than 1, the observation is skipped.

If the WEIGHT statement is used, TABULATE uses the value of the WEIGHT variable to calculate a weighted mean, a weighted variance, and a weighted sum. An observation with a WEIGHT value less than or equal to zero is skipped. Note that the WEIGHT variable value need not be an integer and does not affect the degrees of freedom.

BY Statement

BY *variables;*

A BY statement can be used with PROC TABULATE to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to have been sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

Note that the page dimension expression of a TABLE statement can have an effect similar to that of using a BY statement. The page dimension should be used in most cases where a new page is desired for a given level of a class variable or combination of variables.

Note that your input need not be sorted when the page dimension expression is used.

FORMAT and LABEL Statements

FORMAT *variables format;*
LABEL *variable=label;*

The FORMAT and LABEL statements are especially useful with PROC TABULATE. You can use PROC FORMAT to specify discrete levels for a classification variable. The format specified is also used in any page, row, or column titles where the

class variable appears. Any label text specified in a LABEL statement for a classification or analysis variable is used in any page, row, or column titles where that variable appears.

TABLE Statement

You must be aware of five kinds of operands in the TABLE statement. These operands are essential to the table you are going to design. The operands are

- expressions
- class variables (from the CLASS statement) or ALL
- analysis variables (from the VAR statement)
- statistics
- format specifications.

A TABLE statement consists of one to three dimension expressions separated by commas and followed by an option list. If all three dimensions are specified, the leftmost dimension defines pages, the middle dimension defines rows, and the rightmost dimension defines columns. If two dimensions are specified, the left defines rows and the right defines columns. If a single dimension is specified, it defines columns.

An expression is an alternation of operands and operators. The operators and the effect they produce are listed below:

OPERATOR	ACTION
Comma (,)	Go to a new dimension of table
Asterisk (*)	Subgroup or cross
Blank space	Concatenate tables together
Parentheses ()	Group or specify order
Brackets <>	Specify denominator definitions
Equal =	Assign label to preceding variable or statistic, or complete a format specification

For example, you can design a table analyzing the total sum of units sold by each sales representative. In the TABLE statement, first list the class variable, then the analysis variable, then the statistic. Your PROC TABULATE statements read:

```
PROC TABULATE DATA=SASDATA.SALES;
   CLASS REPNAME;
   VAR NUMBER;
   TABLE REPNAME,NUMBER*SUM;
```

If both crossing and concatenation are used in an expression, the crossing operator takes precedence. Parentheses may be used to group expressions and alter the binding of operators to operands. The form of the TABLE statement is

TABLE {*expression*,} {*expression*,} *expression* {/*options*};

where *expression* is of the form

*expression***expression*	*(Crossing)*
or *expression expression*	*(Concatenation)*
or (*expression*)	*(Grouping)*

Variables and statistics can also be expressions.

classvariable{=*labeltext*}
or *analysisvariable*{=*labeltext*}
or statistic{=*labeltext*}
or *formatspecification*.

The set of analysis variables and the set of statistics can appear in the same dimension or in different dimensions. However, all analysis variables must appear together in one dimension and all statistic names must appear together in one dimension. You cannot cross an analysis variable with another analysis variable or a statistic with another statistic.

If a variable name (classification or analysis) and a statistic name are the same, specify the statistic name in single quotes. The ALL keyword can also be specified in this manner in case of a name conflict. (See **Details** below for a discussion of the keyword ALL.)

A *statistic* in an expression can be a statistic keyword, such as SUM or STD, or a percent expression with the form

```
(PCTN or PCTSUM)<denominatorcrossing..>
```

where *denominator-crossing* is of the form:

```
variable{*variable...}
```

A *formatspecification* is of the form:

```
(F or FORMAT) = (w.d or w. or formatname.)
```

A more elaborate TABLE statement for the last example may be

```
TABLE REPNAME * NUMBER, SUM * F=12. PCTSUM <REPNAME> * F=12.2;
```

where F=12. and F=12.2 are format specifications and <REPNAME> is a denominator crossing.

The following are examples of valid expressions using analysis variable X and class variables A, B, C, and D:

A	class variable
MEAN	statistic
X	analysis variable
A*B	class crossing
A*B*C	multiple class crossing
A*MEAN*X	mean for X within A
A B	class concatenation
A B*C	concatenation of A and B*C
(A B)*C	crossing of A B with C (equivalent to A*C B*C)
(A B)*(C D)*E	crossing of A B with C D which is crossed with E.

If any analysis variables are specified, one or more of the following statistics can be requested:

N	number of observations with nonmissing analysis variable values in the subgroup determined by the combination of class variable values
NMISS	number of observations in the subgroup having missing values for the analysis variable

MEAN	mean
STD	standard deviation
MIN	minimum value
MAX	maximum value
RANGE	range of values
SUM	sum
USS	uncorrected sum of squares
CSS	sum of squares corrected for the mean
STDERR	standard error of the mean
CV	coefficient of variation
T	Student's *t* value for testing the hypothesis that the population mean is 0
PRT	probability of a greater absolute value for the Student's *t* value
VAR	variance
SUMWGT	sum of weight variable values
PCTN	the percentage of frequency (N)
PCTSUM	the percentage of SUM.

If no analysis variables are specified in the TABLE statement, either N or PCTN may be requested as a statistic. If analysis variables are specified, but no statistic, the statistic is SUM. If neither an analysis variable nor a statistic appears in the TABLE statement, each table cell will be the frequency count for a particular interaction of class variable values.

A TABLE statement can define only one table. Multiple TABLE statements can appear after the PROC TABULATE statement, each defining a separate table.

Table Statement Options The options that can appear in the TABLE statement are described below:

MISSTEXT=*'text'*
: supplies up to twenty characters of text to be printed in table cells containing missing values.

FUZZ=*nnn*
: supplies a numeric value against which analysis variable values and table cell values are compared to eliminate trivial values (absolute values less than FUZZ) from computation and printing. A number whose absolute value is less than FUZZ is treated as zero for computations and printing. The default FUZZ value is the smallest representable floating point number on the system on which you are running.

RTSPACE=*n*
RTS=*n*
: supplies an integer value that specifies the number of print positions allotted to row titles. The print positions include those for the left and right boundaries of row titles. (The right row title boundary is also the left boundary of table cells.) The default value is one-fourth of the LINESIZE value.

BOX=_PAGE_
BOX=*variablename*
BOX='*string*'
: specifies the text to be placed in the empty box above the row titles. BOX=_PAGE_ causes the page dimension text to appear in the box. If page dimension text does not fit, it is placed in its normal position and the box left empty. BOX=*variablename* causes the name or label of a variable to appear in the box. BOX='*string*' causes the quoted string to appear in the box. Any name, label, or quoted string that does not fit is truncated.

ROW=FLOAT
ROW=CONSTANT
ROW=CONST
: specifies whether all title elements in a row crossing are allotted space whether blank or not. CONSTANT is default. If ROW=FLOAT and a row title element is blank (for example, N=' ' in the row dimension) the row title space is divided equally among the nonblank title elements in the crossing.

CONDENSE
: requests that multiple logical pages be printed on a single physical page. TABULATE continues to condense the output as long as one or more complete logical pages fits on a single printed page.

KEYLABEL Statement

KEYLABEL *keyword='text';*

where

keyword is one of the valid statistic names discussed above or the universal class variable ALL.

text is up to 40 characters of labeling information. The value of *text* must be enclosed in quotes.

The replacement text is used in any label where the specified keyword is used, unless another label is specified in the TABLE statement. The KEYLABEL statement is useful for relabeling a keyword name once rather than for each occurrence in one or more TABLE statements. Only one keyword label can be specified for each keyword in a particular PROC TABULATE step; if multiple labels are requested for the same keyword, the last one specified is used. An example of a KEYLABEL statement is

```
KEYLABEL ALL='TOTAL $'
         MEAN='AVERAGE'
         PCTSUM='PERCENT OF SUM';
```

DETAILS

Input Data Set

Classification variables can have either alphabetic or numeric values. PROC FORMAT can be used to reduce many input values for a classification variable to a few class levels and to give more descriptive titles to raw level values. Analysis variables must be numeric.

Default Statistics

When an analysis variable but no statistic is specified, the default statistic is SUM. If neither an analysis variable nor a statistic is specified, the default statistic is N, the frequency of occurrence of an interaction of crossed class values.

Missing Values

Missing values may occur in class variables or in analysis variables. A missing value in a class variable causes the observation to be ignored unless MISSING is specified in the PROC statement. The MISSING option causes a missing value to be treated as another classification level; special missing values form other distinct levels.

A missing analysis variable increments the statistic NMISS, the number of observations in the subgroup having a missing value in the analysis variable. The missing value does not affect any other calculations.

Missing table cell values can result from interactions that do not appear in the data (even though individual class values do appear) or from analysis variables that have all missing values. Use the MISSTEXT TABLE statement option to replace missing table cell values with up to twenty characters of text.

Row Title Space

You can use the RTS TABLE statement option to control the space used for the width of row titles. If no RTS value is specified, TABULATE uses one fourth of the linesize. This amount of space is frequently much more than is needed.

The Universal Class Variable ALL

The universal class variable represents a special class that has only one value. When embedded in a crossing, ALL collapses the levels of class variables grouped with it to produce sub-totals and totals. The following table illustrates the hierarchy discussed earlier:

B=1		B=2		B=3	
A=1	A=2	A=1	A=2	A=1	A=2

If B and A are CLASS variables you can summarize all values of A within each value of B with the expression B*(A ALL). The new table follows:

B=1			B=2			B=3		
A=1	A=2	ALL	A=1	A=2	ALL	A=1	A=2	ALL

The additional nodes represent the unions of all A values that share a common B value.

ALL may be specified more than once in an expression. For example, (ALL B)*(ALL A) produces the following table:

ALL			B=1			B=2			B=3		
ALL	A=1	A=2	ALL	A=1	A=2	ALL	A=1	A=2	ALL	A=1	A=2

Consider the marketing research example we have been developing. You can use ALL to produce sales summaries in several dimensions. There are two sets of columns, one for wholesale and one for retail sales. Specifying (SALETYPE ALL) *(QUANTITY AMOUNT) produces a third set of columns that summarize the previous two. There are also two sets of rows, one for REGION and one for CITYSIZE. Each of these sets individually accounts for all sales of a given product. If totalled, each set should yield the same total sales value for each product (unless one of the class variables has missing values). Specifying REGION ALL CITYSIZE ALL produces two summary rows, one for REGION and one for CITYSIZE. There is an individual logical page for each product code. The concatenation, ALL PRODUCT, in the page dimension produces a summary page that incorporates sales for all products. The complete TABLE statement is

```
TABLE ALL PRODUCT, REGION ALL CITYSIZE ALL,
   (SALETYPE ALL)*(QUANTITY*F=6. AMOUNT*F=10.2);
```

ALL

	SALETYPE				ALL	
	R		W			
	QUANTITY	AMOUNT	QUANTITY	AMOUNT	QUANTITY	AMOUNT
	SUM	SUM	SUM	SUM	SUM	SUM
REGION						
NC	3810	95200.00	3810	76200.00	7620	171400.00
NE	4869	121725.00	4869	97380.00	9738	219105.00
SO	5706	143450.00	5706	114120.00	11412	257570.00
WE	5576	139400.00	5576	111520.00	11152	250920.00
ALL	19961	499775.00	19961	399220.00	39922	898995.00
CITYSIZE						
L	9651	241275.00	9651	193020.00	19302	434295.00
M	7400	185950.00	7400	148000.00	14800	333950.00
S	2910	72550.00	2910	58200.00	5820	130750.00
ALL	19961	499775.00	19961	399220.00	39922	898995.00

PRODUCT A100

	SALETYPE				ALL	
	R		W			
	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT
	SUM	SUM	SUM	SUM	SUM	SUM
REGION						
NC	1250	31250.00	1250	25000.00	2500	56250.00
NE	1600	40000.00	1600	32000.00	3200	72000.00
SO	1880	47000.00	1880	37600.00	3760	84600.00
WE	1840	46000.00	1840	36800.00	3680	82800.00
ALL	6570	164250.00	6570	131400.00	13140	295650.00
CITYSIZE						
L	3190	79750.00	3190	63800.00	6380	143550.00
M	2440	61000.00	2440	48800.00	4880	109800.00
S	940	23500.00	940	18800.00	1880	42300.00
ALL	6570	164250.00	6570	131400.00	13140	295650.00

PRODUCT A200

	SALETYPE				ALL	
	R		W			
	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT
	SUM	SUM	SUM	SUM	SUM	SUM
REGION						
NC	1295	32375.00	1295	25900.00	2590	58275.00
NE	1645	41125.00	1645	32900.00	3290	74025.00
SO	1925	48925.00	1925	38500.00	3850	87425.00
WE	1885	47125.00	1885	37700.00	3770	84825.00
ALL	6750	169550.00	6750	135000.00	13500	304550.00
CITYSIZE						
L	3250	81250.00	3250	65000.00	6500	146250.00
M	2500	63500.00	2500	50000.00	5000	113500.00
S	1000	24800.00	1000	20000.00	2000	44800.00
ALL	6750	169550.00	6750	135000.00	13500	304550.00

PRODUCT A300

	SALETYPE					
	R		W		ALL	
	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT	QUANT-ITY	AMOUNT
	SUM	SUM	SUM	SUM	SUM	SUM
REGION						
NC	1265	31575.00	1265	25300.00	2530	56875.00
NE	1624	40600.00	1624	32480.00	3248	73080.00
SO	1901	47525.00	1901	38020.00	3802	85545.00
WE	1851	46275.00	1851	37020.00	3702	83295.00
ALL	6641	165975.00	6641	132820.00	13282	298795.00
CITYSIZE						
L	3211	80275.00	3211	64220.00	6422	144495.00
M	2460	61450.00	2460	49200.00	4920	110650.00
S	970	24250.00	970	19400.00	1940	43650.00
ALL	6641	165975.00	6641	132820.00	13282	298795.00

Percentages: PCTN and PCTSUM

The general form of a percent specification is

(PCTN or PCTSUM) <*denominatordefinitions*> {=*label*} .

PCTN is percentage of frequency (N); it may apply to the frequency of nonmissing analysis variable values or to the frequency of an interaction of class variables. PCTSUM is percentage of SUM; it must apply to the SUM of analysis variable values. Obtain the desired percentages by constructing denominator definitions that collapse appropriate classification levels into totals for the specified analysis variables or interactions.

Denominator definitions The role of the denominator definition is to instruct TABULATE how to collapse classification levels in TABLE statement crossings and how to choose an analysis variable for the denominator. If you understand how ALL is used to summarize classification levels you will understand how denominator definitions work. When you concatenate ALL with a class variable, the result is two crossings. One crossing collects and presents statistics for each level of the class variable; the other crossing collapses the levels of the class variable into a summary or total statistic. When the statistic is N or SUM, ALL represents a total.

You frequently calculate percentages by summing values into a total and then dividing values by the total. TABULATE sums values for this total in the same way that ALL produces totals—by collapsing multiple class levels into a single class level. Denominator definitions tell TABULATE what class levels to collapse. Each denominator definition is a crossing of class variables and possibly a single analysis variable. Multiple definition crossings may appear within brackets. A denominator definition contains a subset of the variables from the TABLE statement crossing. TABULATE adds to the TABLE statement crossing a new crossing in which those class variables named in the denominator definition are replaced by ALL.

For example, **Output 49.8** presents a frequency for each level of B within A and a total frequency of all levels of B within each level of A. And **Output 49.8** presents percentages for each level of B within A.

Output 49.8 Table A*(B ALL)

1

A					
1			2		
B			B		
1	2	ALL	1	2	ALL
N	N	N	N	N	N
10	30	40	20	40	60

Output 49.9 Table A*B*PCTN<B>

1

A			
1		2	
B		B	
1	2	1	2
PCTN	PCTN	PCTN	PCTN
25	75	33	67

The denominators are the sums of B level frequencies within each level of A, that is, the values presented by ALL in the first table. The denominator definition <B> instructs TABULATE to replace B in the TABLE crossing A*B with ALL. The new crossing is A*ALL or simply A. During the data reduction process the new crossing sums the appropriate values into denominator totals.

When there are analysis variables, the denominator totals are N or SUM values for one of them. You can choose any analysis variable by including it in a denominator crossing or you can use each analysis variable in turn by not including any of them in the definition.

The following example presents Y SUM values as percentages of X SUM values.

```
TABLE A*B,C*(X*SUM Y*(SUM PCTSUM<A*B*X>))/RTS=15;
```

There are three class variables. The denominator definition A*B*X instructs TABULATE to collapse the levels of A and B to form a total SUM value of X. The TABLE statement crossing that contains PCTSUM is A*B*C*Y*PCTSUM. A*B is from the row dimension; the rest of the crossing is from the column dimension. The new crossing, constructed as described above, is ALL*ALL*C*X or simply C*X. A and B have been replaced by ALL, and the selected analysis variable is X.

In the following table, the denominator value used when C=1 is 217 and when C=2 is 374. Obtain these values by collapsing all levels of A and B for X; that is, sum X over all levels of A and B. (217 is the total of 59 + 78 + 21 + 59 and 374 is the total of 59 + 14 + 140 + 161.) When A=1 and B=1, the SUM of Y is 123. 123 is 57% of 217.

Output 49.10 Using Y Sum Values as Percentages of X Sum Values

1

		C					
		1			2		
		X	Y		X	Y	
		SUM	SUM	PCTSUM	SUM	SUM	PCTSUM
A	B						
1	1	59	123	57	59	100	27
	2	78	49	23	14	28	7
2	B						
	1	21	71	33	140	106	28
	2	59	49	23	161	2	0

Selecting a definition Each TABLE statement crossing must have a denominator definition to use. It is possible that several definitions in the list apply to a crossing, but only one will yield the correct denominator values. In selecting a definition, TABULATE looks first for a definition that is completely contained in the TABLE statement crossing, such as the class variable and the analysis variable. If no such definition exists, TABULATE looks for a definition whose class variables are all contained in the TABLE statement crossing. A definition that contains only an analysis variable is such a definition—even if the analysis variable is not in the crossing. If no analysis variable is in the definition the analysis variable from the crossing is used.

Note that the order of denominator definitions may affect which definition is chosen. Note also that a definition may be contained in a crossing even if the definition class variables are not contiguous in the crossing, so long as the order of the variables is the same.

Percentages of row totals If all classes in the column dimension and only classes in the column dimension appear in the denominator definition, TABULATE collapses all levels in the column crossing into a denominator total but leaves the row levels uncollapsed. The denominator value is the total for each row. For example,

```
TABLE A,B*(N PCTN<B>);
```

produces the following table.

Output 49.11 Table Producing Percentages of Row Totals

	B			
	1		2	
	N	PCTN	N	PCTN
A				
1	10	25	30	75
2	20	33	40	67

The denominator value for PCTN in row 1 is 40, the total of N values in row 1. The denominator value for PCTN in row 2 is 60, the total of N values in row 2.

Percentages of column totals If all classes in the row dimension and only classes in the row dimension appear in the denominator definition, TABULATE calculates percentages of column totals for each row dimension crossing. For example,

```
TABLE A,B*(N PCTN<A>);
```

produces the table below:

Output 49.12 Table Producing Percentages of Column Totals

	B			
	1		2	
	N	PCTN	N	PCTN
A				
1	10	33	30	43
2	20	67	40	57

The denominator value for PCTN in column 2 is 30, the total of N values in column 1. The denominator value for PCTN in column 4 is 70, the total of N values in column 3.

Percentages of other totals If all classes in the page dimension and only classes in the page dimension appear in the denominator definition, TABULATE calculates percentages of row and column totals for each page dimension crossing.

Denominator definitions may contain class variables from more than one dimension. The percentages calculated are of sub-table totals. For example,

```
TABLE A,B*(N PCTN<A*B>);
```

produces this table:

Output 49.13 Percentages of Row and Column Totals

1

	B			
	1		2	
	N	PCTN	N	PCTN
A				
1	10	10	30	30
2	20	20	40	40

The denominator value for all PCTN cells in this table is 100, the total of the N values in the sub-table of the denominator definition.

Percentages of other analysis variables (a multiple response example) Many applications require the percentage that N or SUM of one analysis variable is of the N or SUM statistic for another analysis variable in the same category. For example, a market research survey may ask for a number of responses to each of several questions. Each observation in this data set contains the following variables:

ID	identification number of respondent
Q	coded question number
V1-V5	five possible responses: a 1 value signifies that response was selected; a missing value indicates no response.

The following statement expresses the frequency of V1-V5 as percentages of the total number of respondents, the frequency of ID. The essential structure of the TABLE statement is

```
TABLE ID*N (V1 V2 V3 V4 V5) * (N PCTN<ID>),Q;
```

Formatting and TABLE statement options have been included below to produce a more attractive table.

```
PROC TABULATE F=7.2;
   CLASS Q;
   VAR ID V1-V5;

   TABLE ID*N='RESPONSES'*F=7. (V1 V2 V3 V4 V5)*(N='COUNT'*F=7.
         PCTN<ID>='PERCENT'),
         Q / RTS=20

   LABEL ID='TOTAL'
         V1='RESPONSE 1'
         V2='RESPONSE 2'
         V3='RESPONSE 3'
         V4='RESPONSE 4'
         V5='RESPONSE 5'
         Q ='QUESTIONS';
```

Output 49.14 Using Formatting and Table Statement Options

		QUESTIONS					
		1	2	3	4	5	6
TOTAL	RESPONSES	300	296	281	286	294	300
RESPONSE 1	COUNT	122	116	108	129	109	110
	PERCENT	40.67	39.19	38.43	45.10	37.07	36.67
RESPONSE 2	COUNT	114	123	111	101	112	113
	PERCENT	38.00	41.55	39.50	35.31	38.10	37.67
RESPONSE 3	COUNT	133	129	107	100	118	128
	PERCENT	44.33	43.58	38.08	34.97	40.14	42.67
RESPONSE 4	COUNT	121	122	127	114	131	114
	PERCENT	40.33	41.22	45.20	39.86	44.56	38.00
RESPONSE 5	COUNT	129	122	111	101	103	123
	PERCENT	43.00	41.22	39.50	35.31	35.03	41.00

Details If no denominator definition list follows the percent keyword, the percentages are calculated on data set totals (or BY-group totals if a BY statement is used).

You can include denominator values in the table by including crossings that collect the same totals. Construct these crossings by substituting ALL for each class variable in the expression crossing that is also in the denominator crossing. For row, column, and page percentages, concatenate ALL in the appropriate dimension, and include ALL as a denominator definition.

The following example illustrates the use of ALL with column percentages. The denominator value is 460, the total of SUM values in column 1.

```
TABLE A*B ALL,X*(SUM PCTSUM<A*B ALL>)/RTS=16;
```

Output 49.15 Using ALL with Column Percentages

1

		X	
		SUM	PCTSUM
A	B		
1	1	160	35
	2	80	17
2	B		
	1	100	22
	2	120	26
ALL		460	100

TABLE statement crossings can be abbreviated if you group elements. For example, A*B*(C D) abbreviates the longer expression (A*B*C A*B*D). However, no grouping is allowed in denominator definitions. Nevertheless, all concatenations that contain a percent keyword must be represented among the denominator definitions for that keyword. In the above example, if the denominators are B*C and B*D, then both crossings must appear in the denominator list, as in

```
TABLE A*B,(C D)*(N PCTN<B*C B*D>);
```

If the denominator is A*B, the crossing needs to be specified only once since it is fully contained in both concatenations, as in this example:

```
TABLE A*B,(C D)*(N PCTN<A*B>);
```

Titling and Formatting

There are four sources of text for page, row, and column titles:

- formatted class values
- literals attached to variable or keyword names in the TABLE statement
- variable names or labels (for both class and analysis variables)
- keyword names or labels (for statistics and ALL).

Formatted class values can be supplied directly from the input data set, or variables can be associated with formats created by the FORMAT procedure. Variable names and keywords are used as titles unless some text has been designated for use in place of the name. Literals may be included in the TABLE statement following the name to which they apply (for example, ALL='REGIONAL TOTAL'). Literals attached to percent keywords must follow the bracketed denominator definition list. Multiple instances of the same name may be attached to different literals. (See ALL in the following example.)

If a LABEL or KEYLABEL statement has been used for a variable or keyword and no literal has been attached to the name in the TABLE statement, then the label text is used in the table titles. Some instances of a name may be attached to a literal in the TABLE statement while others are associated with a label. The appropriate text is used in each case. The syntax of the KEYLABEL statement is identical with that of the LABEL statement.

TABULATE automatically splits title text to fit the space allotted. If a blank or hyphen appears in the title, the split will occur at the blank or hyphen. If there is no blank or hyphen, TABULATE inserts a hyphen at the last position of the current line and continues the title on the next line. If a word in a title is known to be too long to fit in the allotted space, you can force a split by inserting a hyphen into the label or formatted value. For example, if SEPTEMBER is the title of a column 7 or 8 positions wide, you can force a break between M and B by formatting the value as 'SEPTEM-BER'.

Table cell values are formatted using SAS or user defined formats in format items: FORMAT=*formatname.* or F=*formatname.*. Format items appear in the TABLE statement as though they were class or variable names. Format items can appear in any dimension. If two or more format items become crossed with each other, the format with the least scope (that is, the format closer to the column dimension, or closer to the end of the statement) overrides the other formats.

Formats that appear in the page dimension apply to all rows and columns in the associated pages. Formats that appear in the row dimension apply to all columns in the associated rows. Formats that appear in the column dimension apply to all rows in the associated columns. More than one format may apply to rows on the same logical page. Each column width for the logical page is the width of the widest row element in that column. The default format is F=12.2. The following example reworks the marketing research tables used previously:

```
PROC FORMAT;
   VALUE $REGFMT  NC='NORTH CENTRAL'
                  NE='NORTHEAST'
                  SO='SOUTH'
                  WE='WEST';
   VALUE $SIZEFMT  S='UNDER 50000'
                   M='50000 TO 500000'
                   L='OVER 500000';
   VALUE $SALEFMT  W='WHOLESALE'
                   R='RETAIL';

PROC TABULATE;
   CLASS PRODUCT REGION CITYSIZE SALETYPE;
   VAR QUANTITY AMOUNT;
   FORMAT REGION $REGFMT.;
   FORMAT CITYSIZE $SIZEFMT.;
   FORMAT SALETYPE $SALEFMT.;
   LABEL PRODUCT='PRODUCT CODE'
         REGION='REGION OF COUNTRY'
         CITYSIZE='CITY SIZE'
         SALETYPE='TYPE OF SALE'
         AMOUNT='$ AMOUNT';

   TABLE (ALL PRODUCT)*F=8., REGION ALL='REGIONAL TOTAL',
         (SALETYPE ALL)*(QUANTITY AMOUNT);
   KEYLABEL SUM = 'OF SALES'
            ALL = 'TOTAL';
```

Output 49.16 Tables Presenting Sales of a Product

1

TOTAL

	TYPE OF SALE				TOTAL	
	RETAIL		WHOLESALE			
	QUANTITY	$ AMOUNT	QUANTITY	$ AMOUNT	QUANTITY	$ AMOUNT
	OF SALES	OF SALES	OF SALES	OF SALES	OF SALES	OF SALES
REGION OF COUNTRY						
NORTH CENTRAL	3810	95200	3810	76200	7620	171400
NORTHEAST	4869	121725	4869	97380	9738	219105
SOUTH	5706	143450	5706	114120	11412	257570
WEST	5576	139400	5576	111520	11152	250920
REGIONAL TOTAL	19961	499775	19961	399220	39922	898995

2

PRODUCT CODE A100

	TYPE OF SALE					
	RETAIL		WHOLESALE		TOTAL	
	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES
REGION OF COUNTRY						
NORTH CENTRAL	1250	31250	1250	25000	2500	56250
NORTHEAST	1600	40000	1600	32000	3200	72000
SOUTH	1880	47000	1880	37600	3760	84600
WEST	1840	46000	1840	36800	3680	82800
REGIONAL TOTAL	6570	164250	6570	131400	13140	295650

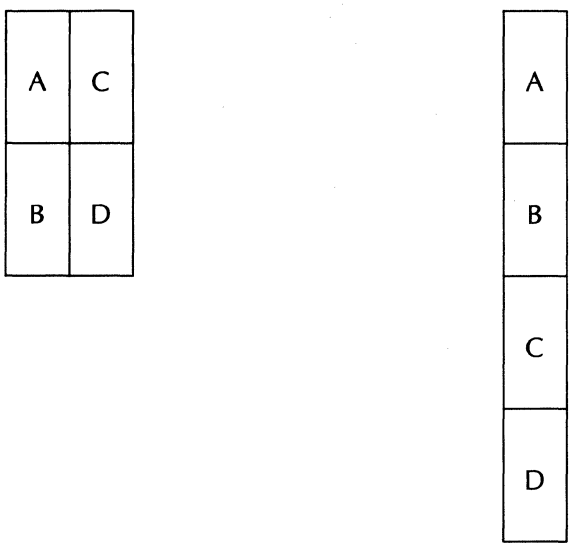
PRODUCT CODE A200

	TYPE OF SALE					
	RETAIL		WHOLESALE		TOTAL	
	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES
REGION OF COUNTRY						
NORTH CENTRAL	1295	32375	1295	25900	2590	58275
NORTHEAST	1645	41125	1645	32900	3290	74025
SOUTH	1925	48925	1925	38500	3850	87425
WEST	1885	47125	1885	37700	3770	84825
REGIONAL TOTAL	6750	169550	6750	135000	13500	304550

PRODUCT CODE A300

	TYPE OF SALE					
	RETAIL		WHOLESALE		TOTAL	
	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES	QUANTITY OF SALES	$ AMOUNT OF SALES
REGION OF COUNTRY						
NORTH CENTRAL	1265	31575	1265	25300	2530	56875
NORTHEAST	1624	40600	1624	32480	3248	73080
SOUTH	1901	47525	1901	38020	3802	85545
WEST	1851	46275	1851	37020	3702	83295
REGIONAL TOTAL	6641	165975	6641	132820	13282	298795

A logical page can contain more rows and columns than can be printed on one physical page. If so, TABULATE automatically divides the logical page into subsets and prints all rows for a given set of columns before moving to the next set of columns. The single logical page depicted below would be divided as indicated:



TABULATE never places multiple logical pages on one physical page. But if a logical page is small, you can force multiple logical pages onto a single physical page with the CONDENSE TABLE statement option.

Limitations

All cell values for a logical page must fit in memory at one time. Increasing the memory available to the SAS system will increase the number of table cells per logical page that can be handled.

Historical Note

TABULATE was inspired in part by the pioneer software package TPL (Table Producing Language) developed at the Bureau of Labor Statistics.

EXAMPLES

The following examples illustrate a number of features of TABULATE. **Example 1** uses several different kinds of formats. **Examples 2** and **3** demonstrate tables containing statistics on readings produced by two hypothetical laboratory machines in each of several labs. This data set contains one observation for each machine for each day of the year. Example 2 contains the values organized into reports on individual machines. Example 3 reorganizes the table to place statistics for each machine in the row dimension.

All examples use FORMAT, LABEL, and KEYLABEL statements to provide descriptive titles. Statistic keyword labels illustrate automatic title splitting at blanks. Example 3 illustrates explicit title splitting using hyphens in the formatted month names. Table cells are formatted in the page dimension except for frequency (N) cells. In each case a format associated with N overrides the page

dimension format. The examples produce summary rows using the universal classifier ALL.

Using Various Formats: Example 1

```
DATA SALES;
   INPUT REGION $ CITYSIZE $ POP PRODUCT $ SALETYPE $ QUANTITY AMOUNT
   ;
   CARDS;
NC S   25000 A100 R 150   3750.00
NE S   37000 A100 R 200   5000.00
SO S   48000 A100 R 410  10250.00
WE S   32000 A100 R 180   4500.00
NC M  125000 A100 R 350   8750.00
NE M  237000 A100 R 600  15000.00
SO M  348000 A100 R 710  17750.00
WE M  432000 A100 R 780  19500.00
NC L  625000 A100 R 750  18750.00
NE L  837000 A100 R 800  20000.00
.
.
.
;



PROC FORMAT;
   VALUE $REGFMT NC='NORTH CENTRAL'
                 NE='NORTHEAST'
                 SO='SOUTH'
                 WE='WEST';
   VALUE $SIZEFMT S='UNDER 50000'
                  M='50000 TO 500000'
                  L='OVER 500000';
   VALUE $SALEFMT R='RETAIL'
                  W='WHOLESALE';

PROC FORMAT;
   PICTURE PCT LOW - <0 =  '000.00%' (PREFIX='-')
               0 - HIGH = '0000.00%';

PROC TABULATE  FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   CLASS REGION SALETYPE;
   VAR   AMOUNT;
   FORMAT REGION $REGFMT.;
   FORMAT CITYSIZE $SIZEFMT.;
   FORMAT SALETYPE $SALEFMT.;

   TABLE ALL*F=DOLLAR11. REGION*F=COMMA11.,    /*ROW DIMENSION*/

      SALETYPE='REGIONAL SALES ANALYSIS'*   /*COLUMN DIMENSION*/
          AMOUNT=' '*
          (SUM    PCTSUM<REGION ALL>*F=PCT.
          N*F=6. PCTN  <REGION ALL>*F=PCT.)
```

```
   /RTSPACE=12;                              /*TABLE OPTIONS*/

KEYLABEL SUM     = 'REVENUE'
         N       = 'LOCA-TIONS'
         PCTSUM  = 'PERCENT OF SALES'
         PCTN    = 'PERCENT OF LOCNS'
         ALL     = 'TOTAL';
```

Output 49.17 Using Various Formats

1

	REGIONAL SALES ANALYSIS							
	RETAIL				WHOLESALE			
	REVENUE	PERCENT OF SALES	LOCA-TIONS	PERCENT OF LOCNS	REVENUE	PERCENT OF SALES	LOCA-TIONS	PERCENT OF LOCNS
TOTAL	$499,775	100.00%	36	100.00%	$399,220	100.00%	36	100.00%
REGION								
NORTH CENTRAL	95,200	19.04%	9	25.00%	76,200	19.08%	9	25.00%
NORTHEAST	121,725	24.35%	9	25.00%	97,380	24.39%	9	25.00%
SOUTH	143,450	28.70%	9	25.00%	114,120	28.58%	9	25.00%
WEST	139,400	27.89%	9	25.00%	111,520	27.93%	9	25.00%

Individual Reports of Machine Readings: Example 2

```
PROC FORMAT;
   VALUE MONFMT  1='JANUARY'
                 2='FEBRUARY'
                 3='MARCH'
                 4='APRIL'
                 5='MAY'
                 6='JUNE'
                 7='JULY'
                 8='AUGUST'
                 9='SEPTEMBER'
                10='OCTOBER'
                11='NOVEMBER'
                12='DECEMBER';

   VALUE MFMT     1='MACHINE 1'
                  2='MACHINE 2';

PROC TABULATE FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   CLASS LAB_ID M_ID MONTHNUM;
   VAR   READING;
   FORMAT MONTHNUM MONFMT.;
   FORMAT M_ID MFMT.;
   LABEL LAB_ID=LABORATORY
         M_ID='MACHINE LAB REPORT'
```

```
      MONTHNUM='MONTH';
KEYLABEL ALL='SUMMARY'
         N='FREQUENCY'
         STD='STANDARD DEVIATION';
TABLE LAB_ID*F=9.3, MONTHNUM ALL, M_ID*READING*
(N*F=9. MEAN RANGE STD) / RTS=12;
```

Output 49.18 Individual Reports of Machine Readings

1

LABORATORY 1

	MACHINE LAB REPORT							
	MACHINE 1				MACHINE 2			
	READING				READING			
	FREQUENCY	MEAN	RANGE	STANDARD DEVIATION	FREQUENCY	MEAN	RANGE	STANDARD DEVIATION
MONTH								
JANUARY	31	0.578	2.616	0.643	31	0.588	2.132	0.493
FEBRUARY	28	0.479	2.460	0.574	28	0.483	1.891	0.427
MARCH	31	0.527	2.215	0.515	31	0.582	2.046	0.502
APRIL	30	0.479	2.677	0.575	30	0.554	2.182	0.518
MAY	31	0.428	1.858	0.414	31	0.458	2.210	0.586
JUNE	30	0.579	2.467	0.589	30	0.339	1.726	0.439
JULY	31	0.529	2.133	0.466	31	0.464	2.353	0.578
AUGUST	31	0.445	2.068	0.453	31	0.528	1.837	0.457
SEPTEMBER	30	0.602	1.933	0.459	30	0.492	1.965	0.464
OCTOBER	31	0.576	2.355	0.627	31	0.474	1.726	0.437
NOVEMBER	30	0.542	2.082	0.532	30	0.413	2.085	0.531
DECEMBER	31	0.610	2.256	0.523	31	0.625	2.052	0.482
SUMMARY	365	0.532	3.000	0.530	365	0.501	2.669	0.495

LABORATORY 2

	MACHINE LAB REPORT							
	MACHINE 1				MACHINE 2			
	READING				READING			
	FREQUENCY	MEAN	RANGE	STANDARD DEVIATION	FREQUENCY	MEAN	RANGE	STANDARD DEVIATION
MONTH								
JANUARY	31	0.532	1.971	0.518	31	0.596	1.550	0.442
FEBRUARY	28	0.467	1.932	0.495	28	0.463	1.819	0.474
MARCH	31	0.410	1.997	0.567	31	0.442	1.537	0.429
APRIL	30	0.487	1.831	0.433	30	0.479	2.352	0.600
MAY	31	0.630	2.042	0.506	31	0.493	1.685	0.432
JUNE	30	0.497	2.468	0.631	30	0.528	1.843	0.510
JULY	31	0.563	1.974	0.495	31	0.688	1.645	0.388
AUGUST	31	0.578	1.800	0.396	31	0.450	2.170	0.446
SEPTEMBER	30	0.367	1.632	0.415	30	0.346	2.154	0.504
OCTOBER	31	0.621	1.735	0.428	31	0.535	2.068	0.451
NOVEMBER	30	0.459	1.773	0.464	30	0.557	1.809	0.468
DECEMBER	31	0.452	2.051	0.546	31	0.420	1.971	0.496
SUMMARY	365	0.506	2.719	0.494	365	0.500	2.566	0.473

Comparative Reports of Machine Readings by Rows: Example 3

This example uses the same machine data as **Example 2** but organizes the report differently.

```
PROC FORMAT;
   VALUE MONFMT 1='JANU-ARY'
                2='FEBRU-ARY'
                3='MARCH'
                4='APRIL'
                5='MAY'
                6='JUNE'
                7='JULY'
                8='AUGUST'
                9='SEPTEMBER'
               10='OCTO-BER'
               11='NOVEM-BER'
               12='DECEM-BER';

   VALUE MFMT   1='1'
                2='2';


PROC TABULATE FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   CLASS LAB_ID M_ID MONTHNUM;
   VAR   READING;
   FORMAT MONTHNUM MONFMT.;
   FORMAT M_ID MFMT.;
```

```
LABEL LAB_ID  ='LABORATORY'
      M_ID    ='MACHINE'
      MONTHNUM='MONTH';
KEYLABEL ALL='SUMMARY'
         N  ='NUMBER OF READINGS'
         STD='STANDARD DEVIATION';
TABLE LAB_ID*F=6.3, (M_ID ALL)*(N*F=6. MEAN RANGE STD),
      READING*(MONTHNUM ALL) / RTS=22;
```

Output 49.19 Comparative Reports of Machine Readings by Rows

1

LABORATORY 1

		READING												
		MONTH												
		JANU-ARY	FEBRU-ARY	MARCH	APRIL	MAY	JUNE	JULY	AUGUST	SEPTE-MBER	OCTO-BER	NOVEM-BER	DECEM-BER	SUMMA-RY
MACHINE														
1	NUMBER OF READINGS	31	28	31	30	31	30	31	31	30	31	30	31	365
	MEAN	0.355	0.407	0.484	0.635	0.560	0.465	0.630	0.464	0.504	0.532	0.518	0.511	0.506
	RANGE	2.445	2.010	2.072	1.988	2.257	1.586	1.776	2.495	1.232	1.398	2.786	2.485	3.210
	STANDARD DEVIATION	0.600	0.494	0.445	0.497	0.532	0.404	0.457	0.474	0.326	0.380	0.623	0.569	0.490
2	NUMBER OF READINGS	31	28	31	30	31	30	31	31	30	31	30	31	365
	MEAN	0.531	0.512	0.576	0.478	0.378	0.595	0.504	0.533	0.419	0.612	0.535	0.512	0.516
	RANGE	2.038	2.167	2.047	2.132	1.853	2.128	2.429	2.648	2.385	1.287	2.622	2.358	3.043
	STANDARD DEVIATION	0.539	0.481	0.485	0.518	0.459	0.499	0.570	0.591	0.544	0.334	0.624	0.578	0.520
SUMMARY	NUMBER OF READINGS	62	56	62	60	62	60	62	62	60	62	60	62	730
	MEAN	0.443	0.459	0.530	0.556	0.469	0.530	0.567	0.499	0.461	0.572	0.526	0.511	0.511
	RANGE	2.869	2.479	2.173	2.441	2.308	2.290	2.429	2.746	2.385	1.398	2.961	2.752	3.292
	STANDARD DEVIATION	0.573	0.486	0.464	0.510	0.501	0.455	0.517	0.533	0.447	0.357	0.618	0.568	0.505

LABORATORY 2

		READING												
		MONTH												
		JANU-ARY	FEBRU-ARY	MARCH	APRIL	MAY	JUNE	JULY	AUGUST	SEPTE-MBER	OCTO-BER	NOVEM-BER	DECEM-BER	SUMMA-RY
MACHINE														
1	NUMBER OF READINGS	31	28	31	30	31	30	31	31	30	31	30	31	365
	MEAN	0.510	0.490	0.594	0.488	0.572	0.498	0.450	0.601	0.685	0.551	0.656	0.545	0.554
	RANGE	1.927	2.592	1.252	2.338	1.583	2.070	1.864	2.348	1.920	2.403	2.399	2.144	2.941
	STANDARD DEVIATION	0.445	0.535	0.334	0.517	0.377	0.468	0.433	0.523	0.518	0.506	0.590	0.490	0.480
2	NUMBER OF READINGS	31	28	31	30	31	30	31	31	30	31	30	31	365
	MEAN	0.472	0.566	0.519	0.679	0.361	0.616	0.560	0.505	0.335	0.507	0.582	0.415	0.509
	RANGE	2.519	1.990	1.755	2.103	1.961	1.718	2.338	2.252	2.197	2.529	1.861	1.873	2.916
	STANDARD DEVIATION	0.607	0.474	0.472	0.522	0.460	0.425	0.543	0.493	0.493	0.603	0.468	0.467	0.507
SUMMARY	NUMBER OF READINGS	62	56	62	60	62	60	62	62	60	62	60	62	730
	MEAN	0.491	0.528	0.557	0.584	0.466	0.557	0.505	0.553	0.510	0.529	0.619	0.480	0.531
	RANGE	2.519	2.592	1.768	2.368	1.961	2.070	2.393	2.470	2.210	2.714	2.399	2.224	3.101
	STANDARD DEVIATION	0.528	0.503	0.408	0.524	0.430	0.448	0.490	0.506	0.532	0.553	0.529	0.479	0.494

Demographic Data Application: Example 4

Data for the following table are selected characteristics of U.S. cities with populations over 50,000 according to *Country and City Data Book, 1983*. The class variables are POPGRP with three levels and REGION with four levels. The TABLE statement requests a table of POPGRP by REGION containing the number of observations, means, and standard deviation for the variable INCOMES, which is specified in the VAR statement.

Another example in "PROC Step Applications" shows the DATA step that reads these census data into a SAS data set. PROC UNIVARIATE is used to determine cutoff points for the POPGRP variable. The SUMMARY and UNIVARIATE procedures are used for other approaches to these data:

```
PROC TABULATE DATA=CITIES;
   TITLE'CENSUS DATA: OUTPUT FROM TABULATE PROCEDURE';
   FORMAT POPGRP PC.;
   VAR INCOME;
   CLASS POPGRP REGION;
   TABLE POPGRP*REGION, INCOME*(N*F=4.0 MEAN STD) / RTSPACE=22;
```

Output 49.20 Demographic Data Application

CENSUS DATA: OUTPUT FROM TABULATE PROCEDURE 38

		PER CAPITA MONEY INCOME (1979)		
		N	MEAN	STD
1980 POPULATION ESTIMATE	REGION			
50,001-75,000	NORTH CENTRAL	46	8255.61	1863.10
	NORTHEAST	33	6903.67	1044.48
	SOUTH	40	7144.55	1263.73
	WEST	43	8527.63	2421.11
75,001-100,000	REGION			
	NORTH CENTRAL	27	7889.59	1520.11
	NORTHEAST	18	6652.89	1743.41
	SOUTH	13	6621.85	1548.09
	WEST	29	8006.28	1637.41
OVER 100,000	REGION			
	NORTH CENTRAL	39	7354.59	941.13
	NORTHEAST	23	6474.70	1197.22
	SOUTH	61	7170.39	1171.39
	WEST	46	8255.61	1180.18

A Marketing Research Application: Example 5

The following marketing research example analyzes advertisement recognition. Each observation contains the sex, income, and yes/no responses of a participant in the study. A yes response is coded as 1, a no response as 0. The DATA step sets the variable NORESP to 1 if all participant responses are missing; otherwise, it sets NORESP to missing. "No" responses are recoded as missing. The first table accumulates the count of nonmissing response values for each ad variable (AD1-AD7) and crosstabulates response frequencies with participant age, sex, and income. The second table crosstabulates income with ad recognition frequencies and calculates percentage of response for each level of income.

Ideas for this example were contributed by The Guttman Group, Inc.

```
PROC FORMAT;
   VALUE AGEFMT 0-29 = 'UNDER 30'
                30-49= '30-49'
                50-70= '50-70'
                OTHER= 'OVER 70';
   VALUE INCFMT 0-9999 = 'UNDER 10,000'
                10000-15000 = '10,000-15,000'
                15001-50000 = 'OVER 15000'
                . = 'NO ANSWER';
   VALUE $SEXFMT 'M'=MALE
                 'F'=FEMALE;

OPTIONS LS=120;
PROC TABULATE F=10. MISSING FORMCHAR='
```

```
TITLE 'AD RECOGNITION STUDY';
CLASS SEX AGE INCOME;
VAR   AD1 AD2 AD3 AD4 AD5 AD6 AD7 NORESP;

FORMAT AGE AGEFMT.;
FORMAT INCOME INCFMT.;
FORMAT SEX $SEXFMT.;

LABEL  AD1 = 'MAN IN CLOWN SUIT'
       AD2 = 'MAN IN ONE RED SHOE'
       AD3 = 'CLOCK FACE'
       AD4 = 'SILHOUETTE'
       AD5 = 'SAW SOMETHING IN MAGAZINE'
       AD6 = 'SAW SOMETHING ON TV'
       AD7 = 'SAW SOMETHING IN NEWSPAPER'
       NORESP = 'NO RESPONSE';

TABLE ALL='----   TOTAL   ----'           /*ROW DIMENSION*/
      AD1 AD2 AD3 AD4 AD5 AD6 AD7 NORESP,

      (ALL                                /*COLUMN DIMENSION*/
       SEX='+---S E X---+'
       AGE='+----------A G E----------+'
       INCOME='+-------I N C O M E-------+')
       *N='------'*F=6.
       / RTS=30
         BOX='NUMBER OF NON-MISSING RESPONSES FOR EACH AD';

TABLE ALL*(N PCTN) INCOME*PCTN,
      ALL AD1 AD2 AD3 AD4 AD5 AD6 AD7
      / RTS=18 ROW=FLOAT
        BOX='RESPONSE PERCENTAGES FOR INCOME LEVELS';

KEYLABEL N='COUNT'
         ALL='TOTAL'
         PCTN='PERCENT';
```

Output 49.21 A Marketing Research Application

AD RECOGNITION STUDY 1

NUMBER OF NON-MISSING RESPONSES FOR EACH AD	TOTAL	SEX: FEMALE	SEX: MALE	AGE: UNDER 30	AGE: 30-49	AGE: 50-70	AGE: OVER 70	INCOME: NO ANSWER	INCOME: UNDER 10,000	INCOME: 10,000 15,000	INCOME: OVER 15000
---- TOTAL ----	100	53	47	34	26	15	25	11	21	11	57
MAN IN CLOWN SUIT	34	18	16	18	5	5	6	3	7	4	20
MAN IN ONE RED SHOE	62	25	37	21	16	10	15	6	15	6	35
CLOCK FACE	31	15	16	9	5	5	12	2	8	2	19
SILHOUETTE	27	11	16	7	6	5	9	3	5	3	16
SAW SOMETHING IN MAGAZINE	44	28	16	17	14	3	10	7	7	6	24
SAW SOMETHING ON TV	28	14	14	7	8	4	9	3	7	1	17
SAW SOMETHING IN NEWSPAPER	19	13	6	6	5	3	5	2	6	3	8
NO RESPONSE	2	2	0	2	0	0	0	0	1	1	0

AD RECOGNITION STUDY 2

RESPONSE PERCENTAGES FOR INCOME LEVELS		TOTAL	MAN IN CLOWN SUIT	MAN IN ONE RED SHOE	CLOCK FACE	SILHOUETTE	SAW SOMETHING IN MAGAZINE	SAW SOMETHING ON TV	SAW SOMETHING IN NEWSPAPER
TOTAL	COUNT	100	34	62	31	27	44	28	19
	PERCENT	100	100	100	100	100	100	100	100
INCOME									
NO ANSWER	PERCENT	11	9	10	6	11	16	11	11
UNDER 10,000	PERCENT	21	21	24	26	19	16	25	32
10,000- 15,000	PERCENT	11	12	10	6	11	14	4	16
OVER 15000	PERCENT	57	59	56	61	59	55	61	42

A Detailed Budget Summary: Example 6

This example uses a DATA step to calculate row totals and PROC TABULATE to display the report with column sub-totals and totals.

```
PROC FORMAT;
   VALUE DEPT 1='ACCOUNTING'
              2='HUMAN RESOURCES'
              3='SYSTEMS'
              4='PRODUCTION'
              5='MARKETING' ;

DATA TABULATE;
   INPUT DEPT ACCT JAN FEB MAR BUDGET;
```

```
   TOTAL=JAN + FEB + MAR;
   LEFT=BUDGET - TOTAL;
CARDS;
01  01345  12980  14009  17800  40000
01  01578   8000   7900   4500  40000
01  01674  11950  13534  17994  40000
02  02134  34520  26560  24399  80000
02  02403  10435  15494  10009  40000
03  04138  24850  22530  24399  70000
03  04279   9984  14209  13500  40000
03  04290  10948  14539  11459  40000
04  05139  12000  14532  12098  40000
04  05260  15893  14099   7304  40000
04  05370  11980  13900  11480  40000
04  05399  20435  19095  18053  60000
05  06120  23435  23543  19054  60000
05  06342  13049  15349  18943  50000
05  06401  20943  25943  19432  65000
;

PROC TABULATE FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   TITLE1 '  ';
   TITLE2 '  ';
   TITLE3 'FLEET FOOTWEAR, INC.';
   TITLE4 'DEPARTMENTAL BUDGETARY REPORT';
   TITLE5 'FOR THE FIRST QUARTER';
   TITLE6 '  ';
   CLASS DEPT ACCT;
   VAR JAN FEB MAR TOTAL LEFT BUDGET;
   FORMAT DEPT DEPT.;
   LABEL TOTAL='QUARTER TOTAL';
   LABEL BUDGET='ALLOCATED FUNDS';
   LABEL LEFT='REMAINING FUNDS';
   LABEL JAN='JANUARY EXPENDITURES';
   LABEL FEB='FEBRUARY EXPENDITURES';
   LABEL MAR='MARCH EXPENDITURES';

   TABLE /*---ROW DIMENSION---*/
        (DEPT=' '*(ACCT=' ' ALL='SUB-TOTAL') ALL='TOTAL')
        *F=COMMA12.2,

        /*---COLUMN DIMENSION---*/
        ((JAN FEB MAR)*SUM=' '
        TOTAL*SUM=' '
        (BUDGET LEFT)*SUM=' ') /

        /*---TABLE OPTIONS---*/
        BOX='DEPARTMENT      ACCOUNT';
```

Output 49.22 A Detailed Budget Summary

1

FLEET FOOTWEAR, INC.
DEPARTMENTAL BUDGETARY REPORT
FOR THE FIRST QUARTER

DEPARTMENT	ACCOUNT	JANUARY EXPENDITURES	FEBRUARY EXPENDITURES	MARCH EXPENDITURES	QUARTER TOTAL	ALLOCATED FUNDS	REMAINING FUNDS
ACCOUNTING	1345	12,980.00	14,009.00	17,800.00	44,789.00	40,000.00	-4,789.00
	1578	8,000.00	7,900.00	4,500.00	20,400.00	40,000.00	19,600.00
	1674	11,950.00	13,534.00	17,994.00	43,478.00	40,000.00	-3,478.00
	SUB-TOTAL	32,930.00	35,443.00	40,294.00	108,667.00	120,000.00	11,333.00
HUMAN RESOURCES	2134	34,520.00	26,560.00	24,399.00	85,479.00	80,000.00	-5,479.00
	2403	10,435.00	15,494.00	10,009.00	35,938.00	40,000.00	4,062.00
	SUB-TOTAL	44,955.00	42,054.00	34,408.00	121,417.00	120,000.00	-1,417.00
SYSTEMS	4138	24,850.00	22,530.00	24,399.00	71,779.00	70,000.00	-1,779.00
	4279	9,984.00	14,209.00	13,500.00	37,693.00	40,000.00	2,307.00
	4290	10,948.00	14,539.00	11,459.00	36,946.00	40,000.00	3,054.00
	SUB-TOTAL	45,782.00	51,278.00	49,358.00	146,418.00	150,000.00	3,582.00
PRODUCTION	5139	12,000.00	14,532.00	12,098.00	38,630.00	40,000.00	1,370.00
	5260	15,893.00	14,099.00	7,304.00	37,296.00	40,000.00	2,704.00
	5370	11,980.00	13,900.00	11,480.00	37,360.00	40,000.00	2,640.00
	5399	20,435.00	19,095.00	18,053.00	57,583.00	60,000.00	2,417.00
	SUB-TOTAL	60,308.00	61,626.00	48,935.00	170,869.00	180,000.00	9,131.00
MARKETING	6120	23,435.00	23,543.00	19,054.00	66,032.00	60,000.00	-6,032.00
	6342	13,049.00	15,349.00	18,943.00	47,341.00	50,000.00	2,659.00
	6401	20,943.00	25,943.00	19,432.00	66,318.00	65,000.00	-1,318.00
	SUB-TOTAL	57,427.00	64,835.00	57,429.00	179,691.00	175,000.00	-4,691.00
TOTAL		241,402.00	255,236.00	230,424.00	727,062.00	745,000.00	17,938.00

Medical Events by Age Group: Example 7

This example displays row, column, and data set percentages in the same table. Data are for a hypothetical survey of medical events in a study population. Each observation contains a numeric subject id, the age of the subject, and a medical event code. On the table, % OF THIS EVENT shows how each event is distributed across age groups, a row percent. % OF AGE GROUP shows how each age group is distributed across events, a column percent. % OF ALL EVENTS shows how the whole population is distributed across events and age groups, a data set percentage.

```
PROC FORMAT;
   VALUE AGEFMT 0   - 20 = '30 - 35'
                21 - 50 = '36 - 40'
                51 - 70 = '41 - 45'
                71 -100 = '46 - 50';
   VALUE CDFMT  0   - 50 = 'CHEST PAIN'
                51 - 75 = 'HOSPITAL VISIT'
```

```
                76 - 90 = 'HEART ATTACK'
                91 -100 = 'MORTALITY';

PROC TABULATE F=7.2 FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   CLASS AGE CODE;
   VAR ID;

   TABLE
         /*---ROW DIMENSION---*/
         (ALL CODE=' ')*
            ( N*F=5.
             (PCTN<ID AGE>='% OF THIS EVENT'
              PCTN<CODE ALL>='% OF AGE GROUP'
              PCTN<ALL*AGE*ID ALL*ID CODE*AGE*ID CODE*ID>
                             ='% OF ALL EVENTS' )*F=7.2),

         /*---COLUMN DIMENSION---*/
         ID AGE

         /*---TABLE OPTIONS---*/
         / RTS=33 MISSTEXT='NONE'
           BOX='MEDICAL EVENTS BY AGE GROUP';

   LABEL ID = 'ALL AGE GROUPS'
         AGE = 'AGE GROUPS';

   FORMAT AGE AGEFMT.;
   FORMAT CODE CDFMT.;

   KEYLABEL ALL = 'ALL EVENTS'
            N = 'FREQUENCY';
```

Output 49.23 Row, Column, and Data Set Percents

MEDICAL EVENTS BY AGE GROUP		ALL AGE GROUPS	AGE GROUPS			
			30 - 35	36 - 40	41 - 45	46 - 50
ALL EVENTS	FREQUENCY	100	15	35	21	29
	% OF THIS EVENT	100.00	15.00	35.00	21.00	29.00
	% OF AGE GROUP	100.00	100.00	100.00	100.00	100.00
	% OF ALL EVENTS	100.00	15.00	35.00	21.00	29.00
CHEST PAIN	FREQUENCY	50	6	19	10	15
	% OF THIS EVENT	100.00	12.00	38.00	20.00	30.00
	% OF AGE GROUP	50.00	40.00	54.29	47.62	51.72
	% OF ALL EVENTS	50.00	6.00	19.00	10.00	15.00
HOSPITAL VISIT	FREQUENCY	20	3	5	6	6
	% OF THIS EVENT	100.00	15.00	25.00	30.00	30.00
	% OF AGE GROUP	20.00	20.00	14.29	28.57	20.69
	% OF ALL EVENTS	20.00	3.00	5.00	6.00	6.00
HEART ATTACK	FREQUENCY	24	4	8	5	7
	% OF THIS EVENT	100.00	16.67	33.33	20.83	29.17
	% OF AGE GROUP	24.00	26.67	22.86	23.81	24.14
	% OF ALL EVENTS	24.00	4.00	8.00	5.00	7.00
MORTALITY	FREQUENCY	6	2	3	NONE	1
	% OF THIS EVENT	100.00	33.33	50.00	NONE	16.67
	% OF AGE GROUP	6.00	13.33	8.57	NONE	3.45
	% OF ALL EVENTS	6.00	2.00	3.00	NONE	1.00

A Multiple Response Survey: Example 8

In many cases you can use TABULATE to analyze multiple response data. In this example the survey questions are:

1. Which cereal do you buy?
2. Which of the following reasons made you choose this cereal? (Circle all that apply.)

   Flavor
   Texture
   Nutritional value
   Price
   Availability.

Each observation contains a respondent id, a cereal code, and five yes/no (1/missing) response values. Percentages in the table are percentages of respondents for each cereal. You can easily obtain percentages of total respondents as well.

```
PROC FORMAT;
   VALUE CFMT 1 = 'CRUNCHY NUGGETS'
              2 = 'BREAK-FAST FANTASY'
```

```
                3 = 'SUNRISE SERE-NADE'
                4 = 'SUGAR BOWL'
                5 = 'FULL-O-FIBER';


PROC TABULATE F=7.2 FORMCHAR='FABFACCCBCEB8FECABCBBB'X;
   CLASS CEREAL;
   VAR ID R1-R5;
   TABLE ID*N='RESPONDENTS'*F=7.
         (R1 R2 R3 R4 R5)*(N='COUNT'*F=7. PCTN<ID>='PERCENT'),
         CEREAL=' ' ALL='ALL CEREALS'
         / RTS=27 BOX='CEREAL SURVEY';
   LABEL ID='TOTAL'
         R1='FLAVOR'
         R2='TEXTURE'
         R3='NUTRITIONAL VALUE'
         R4='PRICE'
         R5='AVAILABIL-ITY';
   FORMAT CEREAL CFMT.;
```

Output 49.24 Multiple Response Survey

1

CEREAL SURVEY		CRUNCHY NUGGETS	BREAK-FAST FANTASY	SUNRISE SERE-NADE	SUGAR BOWL	FULL-O-FIBER	ALL CEREALS
TOTAL	RESPONDENTS	270	322	312	267	329	1500
FLAVOR	COUNT	215	266	247	215	255	1198
	PERCENT	79.63	82.61	79.17	80.52	77.51	79.87
TEXTURE	COUNT	212	239	228	198	249	1126
	PERCENT	78.52	74.22	73.08	74.16	75.68	75.07
NUTRITIONAL VALUE	COUNT	173	206	204	161	224	968
	PERCENT	64.07	63.98	65.38	60.30	68.09	64.53
PRICE	COUNT	132	163	160	134	159	748
	PERCENT	48.89	50.62	51.28	50.19	48.33	49.87
AVAILABIL-ITY	COUNT	108	127	122	104	129	590
	PERCENT	40.00	39.44	39.10	38.95	39.21	39.33

REFERENCES

County and City Data Book, 1983 Files on Tape (machine-readable data file)/ prepared by the Bureau of the Census. Washington: The Bureau (producer and distributor), 1984.

County and City Data Book, 1983 Files on Tape Technical Documentation/ prepared by the Data User Services Division, Bureau of the Census. Washington: The Bureau, 1984.

Chapter 50

The TAPECOPY Procedure

Operating systems: CMS and OS

ABSTRACT

The TAPECOPY procedure copies an entire tape volume (tape) or files from one or several tape volumes to one output tape volume. PROC TAPECOPY always begins writing at the beginning of the output tape volume; any files that exist on the output tape prior to the copy operation are destroyed. The procedure works under OS and CMS. For the sake of clarity the procedure is described in separate sections for each system: first for OS, and then for CMS. Examples for PROC TAPECOPY under both systems are at the end of the chapter.

INTRODUCTION TO PROC TAPECOPY UNDER OS

The TAPECOPY procedure can copy standard-labeled or nonlabeled 9-track tapes. You can specify, within limits, whether the output tape is standard labeled (SL) or nonlabeled (NL). You cannot create an SL tape using an NL input tape because TAPECOPY cannot manufacture tape labels. Nor can you change an output tape volume for which LABEL=(,SL) is specified in a DD statement into a nonlabeled tape. PROC TAPECOPY does allow you to write over an existing volume label on a standard-labeled tape if you specify LABEL=(,BLP) in the DD statement.

The JCL DD statement parameter LABEL=(,BLP) must be authorized specifically by each computing installation. If your installation allows the BLP specification, ANSI-labeled, non-standard-labeled, and standard-user-labeled tapes can be treated as nonlabeled tape volumes. If the BLP specification is not authorized at your installation, LABEL=(,BLP) is treated as LABEL=(,NL). PROC TAPECOPY will work as you expect if your tape is in fact nonlabeled; otherwise, the operating system does not allow TAPECOPY to use the tape, thus preserving the label.

References to specifying LABEL=(,BLP) throughout this OS description assume that LABEL=(,BLP) is a valid specification at your installation.

SPECIFICATIONS UNDER OS

The following statements are used in the TAPECOPY procedure:

PROC TAPECOPY *options*;
  INVOL *options*;
  FILES *filenumbers*;

PROC TAPECOPY Statement

PROC TAPECOPY *options*;

The options listed below can appear in the PROC TAPECOPY statement:

NOLIST
: suppresses printing of the tape characteristics and a summary of copied files. Whether NOLIST is specified or not, the SAS log contains a brief summary of PROC TAPECOPY's action; this summary is usually enough to verify proper functioning of PROC TAPECOPY if you are familiar with the contents of the input tape(s).

DEN=*density*
: specifies the *density* of the output tape. If the DEN= option appears in the PROC TAPECOPY statement, it overrides any DCB=DEN specification in the DD statement for the output tape volume. If you do not specify a density in the PROC TAPECOPY statement or in the DD statement, the operating system writes the tape at its default density, usually the highest density at which the unit allocated to the output tape volume can record.

Valid density values are

DEN=2 DEN=800	800 bpi
DEN=3 DEN=1600	1600 bpi
DEN=4 DEN=6250	6250 bpi

LABEL=SL
LABEL=NL
: specifies whether the output tape volume is to be standard labeled (LABEL=SL) or nonlabeled (LABEL=NL).

  Be careful not to confuse the LABEL= option on the PROC statement with the DD statement parameter LABEL=(,*specification*). The PROC statement LABEL= option specifies whether the output tape is standard labeled or nonlabeled **after** the copy operation. The output tape volume's DD statement LABEL= parameter specifies what the output tape's label status is **before** the copy operation.

  The DD statement for nonlabeled output tapes must specify LABEL=(,NL) or LABEL=(,BLP). If the output tape has an existing label (before the copy operation) and the output tape is to be nonlabeled (after the copy operation), the DD statement **must** specify LABEL=(,BLP).

  The default LABEL= option value is NL when multiple input volumes are used and the DD statements for any of them specify LABEL=(,NL). If there are multiple input tapes and LABEL=(,NL) is **not** specified for any of them and if the first input tape volume is actually standard labeled, then the default LABEL= option value is SL. This holds even if the DD statement specifies LABEL=(,BLP) for the first tape; in this case, PROC TAPECOPY reads the tape volume's first record to determine the actual label type.

COPYVOLSER
: specifies that the output tape should have a standard label with the same volume serial as the first input tape.
  COPYVOLSER is only effective when:

  - the output tape volume is to be standard labeled, that is, LABEL=SL, and
  - the output tape DD statement specifies LABEL=(,NL) or LABEL=(,BLP).

  When these conditions are not met, PROC TAPECOPY stops processing.

INDD=*fileref*
: specifies the *fileref* of the JCL DD statement describing the first input tape volume. The default INDD= value is VOLIN.

OUTDD=*fileref*
: specifies the *fileref* of the JCL DD statement describing the output tape. The default OUTDD= value is VOLOUT.

INVOL=*volume serial*
: specifies the *volume serial* of the first input tape when deferred mounting is specified in the DD statement for the first input tape. The INVOL= option specification overrides the volume serial, if any, specified in the DD statement for the tape.

  You should not specify the INVOL= option unless you are using deferred mounting.

OUTVOL=*volume serial*
: specifies the *volume serial* of the output tape when deferred mounting is specified in the DD statement for the output tape. The OUTVOL= option specification overrides the volume serial, if any, specified on the DD statement for the tape.

  You should not specify the OUTVOL= option unless you are using deferred mounting.

NORER
: requests that the "reduced error recovery for tape devices" feature of the operating system not be specified for each input tape volume. Some tapes of marginal quality can be read successfully by PROC TAPECOPY when NORER is specified since the error recovery procedures are more exhaustive.

NEWVOLSER=*new volume serial*
: specifies a *new volume serial* for the output tape. NEWVOLSER is effective only if the output tape is to be standard labeled. If the output tape has an existing label, the DD statement for the output tape must specify LABEL=(,BLP); otherwise, PROC TAPECOPY stops processing and does not write over the label.

NOFSNRESEQ
NFR
: specifies that file sequence numbers in the file labels should not be resequenced when a standard-labeled output tape volume is being produced. PROC TAPECOPY normally resequences these numbers and updates the label to reflect the ordinal position of the file on the output tape as it is copied and the actual density at which the output tape is written.

INVOL Statement

INVOL options;

The INVOL statement defines an input tape volume from which some or all files are to be copied to the output tape volume. The INVOL statement is not necessary if you are using only one input tape or for the first of several input tapes (use the INDD= and INVOL= options of the PROC TAPECOPY statement instead). When you are using several input tapes, use an INVOL statement for each tape after the first input tape.

The options below can be used in the INVOL statement:

INDD=*fileref*
: specifies the *fileref* of the JCL DD statement describing the current input tape. The default INDD= value is the *fileref* already in effect for the previous input tape volume, as specified in the PROC TAPECOPY statement or in the last INVOL statement.

INVOL=*volumeserial*
: specifies the *volume serial* of the current input tape. Use the INVOL= option when the JCL DD statement for the input tape specifies deferred mounting (as described above in **PROC TAPECOPY Statement**), or when you are reusing a DD statement (and tape drive); that is, the *fileref* is the same, but you want a different tape volume on the same unit.

NL
: specifies that the input tape is nonlabeled; if LABEL=(,SL) or LABEL=(,BLP) has been specified in the DD statement for the input tape and the tape is actually standard-labeled, specifying the NL option causes the tape to be treated as if it were nonlabeled. In this case, any file numbers specified in FILES statements must be physical file numbers, not logical file numbers.

SL
: specifies that the input tape is standard labeled. If you specify LABEL=(,BLP) in the DD statement for the input tape and specify SL in the INVOL statement, PROC TAPECOPY verifies that the tape is standard labeled. Do not specify SL unless the tape is actually standard labeled.

  If you do not specify NL or SL in the INVOL statement, the actual input tape label type determines whether PROC TAPECOPY treats the tape as NL or SL, even when LABEL=(,BLP) is specified in the DD statement.

DSNAME='*data set name*'
DSN='*data set name*'
: specifies the *data set name* of the first file of the current input tape. You **must** use this option when:

  - the data set name specified in the DD statement is incorrect or missing, and
  - LABEL=(,SL) is specified (or implied by default) on the input tape volume DD statement.

  You typically use this option when:

  - the DD statement for the input tape specifies deferred mounting, or
  - you are reusing a DD statement (and tape drive); that is, when the *fileref* is the same but you want another standard-labeled tape volume on the same unit. LABEL=(,SL) should be specified or defaulted, and the data set name cannot be the same as that on the previous tape used with this *fileref*.

NORER
: requests that the "reduced error recovery for tape devices" feature of the operating system not be specified for the input tape volume. Some tapes of marginal quality can be read successfully by PROC TAPECOPY when this option is specified since the error recovery procedures are more exhaustive. If NORER is specified in the PROC TAPECOPY statement, then NORER is in effect for all input tape volumes and INVOL statements.

FILES Statement

FILES *filenumbers*;
FILE *filenumbers*;

When you want to copy certain files from an input tape, use the FILES statement to specify the files you want to copy. Use as many FILES statements as you want. Give the physical file numbers for nonlabeled tapes or labeled tapes being treated as nonlabeled. Give the logical file numbers for standard-labeled tapes not being treated as nonlabeled, even when the output tape volume is to be nonlabeled (LABEL=NL).

You can specify a range of files by putting a dash between two file numbers:

```
PROC TAPECOPY;
   FILES 1-7;
```

In a range, the second number must be greater than the first. The keyword EOV can be used as the second file in a range; PROC TAPECOPY copies all files on the input tape until the end of the volume (in most cases, a double tapemark). On a nonlabeled tape, you can copy files beyond the double tapemark by specifying the physical file number, counting tapemarks as usual. If another double tapemark exists on the input tape volume, you can then specify EOV in another range.

File numbers in a FILES statement can be specified in any order; you might want to copy file 5 and then file 2 and then file 1.

```
PROC TAPECOPY;
   FILES 5 2;
   FILE  1;
```

If you are using only one input tape, the FILES statement(s) can directly follow the PROC TAPECOPY statement. When you use several input tape volumes, follow each INVOL statement with the FILES statement(s).

DETAILS UNDER OS

Usage Notes

Input tape DD statement requirements In the DD statement describing an input tape, you need to specify only UNIT, VOL=SER, DISP, and usually either LABEL or DSN. The VOL=SER value gives the volume serial of the first input tape; VOL=SER can be omitted if the UNIT parameter specifies deferred mounting; for example, UNIT=(*tape*,,DEFER). (If you specify deferred mounting, remember to use the INVOL= option in the PROC statement or INVOL statement(s) to specify the volume serial of the input tape.)

For a nonlabeled input tape, you must specify LABEL=(,NL) or LABEL=(,BLP) in the DD statement. If you are unsure whether the input tape volume is labeled or nonlabeled, specify LABEL= (,BLP) in the input tape DD statement, if your installation allows it.

For a standard-labeled input tape at an installation that does not allow LABEL=(,BLP), specify LABEL=(,SL) and the DSN parameter, giving the DSNAME of the first data set on the tape.

Output tape DD statement requirements In the DD statement describing the output tape, you usually need to specify only the UNIT, VOL=SER, DISP, and possibly LABEL or DSN parameters. The VOL=SER value gives the volume serial of the output tape. VOL=SER can be omitted if the UNIT parameter specifies deferred mounting; for example, UNIT=(*tape*,,DEFER). (If you specify deferred mounting, use the OUTVOL= option in the PROC statement to specify the volume serial of the output tape.)

You should normally specify DISP=(NEW,KEEP) for the output tape in the DD statement. At some installations it may be necessary to specify DISP=(OLD,KEEP)

along with the DSN parameter, giving the DSNAME of the first data set on the tape volume. The LABEL parameter should give the tape's label type before the TAPECOPY procedure is executed, regardless of its label type after the copying operation.

Printed Output

The TAPECOPY procedure prints a listing of the input and output tape characteristics and a summary of the files copied on the SAS log.

INTRODUCTION TO PROC TAPECOPY UNDER CMS

The TAPECOPY procedure can copy standard-labeled or nonlabeled 9-track tapes under CMS. You can specify, within limits, whether the output tape is standard labeled (SL) or nonlabeled (NL). You cannot create an SL tape using an NL input tape because PROC TAPECOPY cannot manufacture tape labels. You can create a nonlabeled output tape volume from a labeled input tape. Under CMS, TAPECOPY writes over any existing labels on the output tape.

It is not possible to use tapes (and therefore, PROC TAPECOPY) at all CMS sites. Some CMS sites do not have any CMS tape mount commands, but they allow you to mount tapes by contacting the computer operator. If your installation does not support CMS tape mount commands, you can use PROC TAPECOPY, but you cannot specify deferred tape mounting for input tapes. If your installation does have CMS tape mount commands, you can specify deferred tape mounting with an option in the procedure's INVOL statement (see below).

Note: PROC TAPECOPY supports deferred mounting of input tapes only.

SPECIFICATIONS UNDER CMS

The following statements are used in PROC TAPECOPY:

PROC TAPECOPY *options*;
INVOL *options '/'CMSmount'*;
FILES *filenumbers*;

PROC TAPECOPY Statement

PROC TAPECOPY *options*;

The options listed below can appear in the PROC TAPECOPY statement:

NOLIST
: suppresses printing of the tape characteristics and a summary of copied files. Whether NOLIST is specified or not, the SAS log contains a brief summary of PROC TAPECOPY's action; this summary is usually enough to verify proper functioning of TAPECOPY if you are familiar with the contents of the input tape(s).

DEN=*density*
: specifies the density of the output tape. If the DEN= option appears in the PROC TAPECOPY statement, it overrides any density specification in the FILEDEF for the output tape volume. If you do not specify a density in the PROC TAPECOPY statement, the tape is written at the highest density possible.

Valid density values are

DEN=2 DEN=800	800 bpi
DEN=3 DEN=1600	1600 bpi
DEN=4 DEN=6250	6250 bpi

LABEL=SL
LABEL=NL
LABEL=BLP
: specifies whether the output tape volume is to be standard-labeled (LABEL=SL) or nonlabeled (LABEL=NL) or that all label processing should be bypassed (LABEL=BLP). The default LABEL= option value is BLP. Do not specify SL if you intend to copy any nonlabeled tapes.

  If SL is specified, the output tape is SL; if NL is specified, the output tape is NL. When BLP is specified, all files are treated as physical files and the distinction between data files and label files is irrelevant; therefore, the output tape has the label status of the input tape.

COPYVOLSER
: specifies that the output tape should have a standard label with the same volume serial as the first input tape.

  COPYVOLSER is only effective when the output tape volume is to be standard labeled, that is, LABEL=SL. When this condition is not met, PROC TAPECOPY stops processing.

INDD=*fileref*
: specifies the *fileref* of the FILEDEF describing the first input tape volume. The default INDD= value is VOLIN.

TAP1
: specifies tape drive 181 as the input tape device. There is no need for an input tape FILEDEF if this option is used. If INDD= is not specified and there is no FILEDEF for VOLIN, TAP1 is the default. Do not use both INDD= and TAP1.

OUTDD=*fileref*
: specifies the *fileref* of the FILEDEF describing the output tape. The default OUTDD= value is VOLOUT.

TAP*n*
: specifies tape drive 18*n* as the output tape device, where *n* is 2, 3, or 4. If TAP*n* is specified, there is no need for an output tape FILEDEF. If OUTDD= has not been specified and there is no FILEDEF for VOLOUT, TAP*n* defaults to TAP2. Do not specify both INDD= and TAP*n*.

INVOL=*volume serial*
: specifies the *volume serial* of the first input tape.

NEWVOLSER=*new volume serial*
: specifies a *new volume serial* for the output tape. NEWVOLSER is effective only if the output tape is to be standard labeled.

DETACH
: requests that all tape drives used by PROC TAPECOPY be detached after the procedure has executed.

NOFSNRESEQ
NFR
: specifies that file sequence numbers in the file labels should not be resequenced when a standard-labeled output tape volume is being produced. PROC TAPECOPY normally resequences these numbers and updates the label to reflect the ordinal position of the file on the output tape as it is copied and the actual density at which the output tape is written.

INVOL Statement

INVOL options */'CMSmount'*;

The INVOL statement defines an input tape volume from which some or all files are to be copied to the output tape volume. The INVOL statement is not necessary if you are using only one input tape or for the first of several input tapes (use the INDD= and INVOL= options of the PROC TAPECOPY statement instead). When you are using several input tapes, use an INVOL statement for each tape after the first input tape.

If you want to use deferred mounting for an input tape, you must use an INVOL statement with the */'CMSmount'* option (below). (You cannot use deferred mounting for an output tape.)

The options below can be used in the INVOL statement:

INDD=*fileref*
: specifies the *fileref* of the FILEDEF describing the current input tape. The default INDD= value is the *fileref* already in effect for the previous input tape volume, as specified in the PROC TAPECOPY statement or the last INVOL statement.

TAP*n*
: specifies the tape drive to use, where *n* is 1, 2, 3, or 4. The default value is the TAP*n* in effect from the PROC statement or previous INVOL statement. Do not use both INDD= and TAP*n*.

INVOL=*volume serial*
: specifies the *volume serial* of the current input tape.

NL
: specifies that the input tape is nonlabeled; if the input tape is actually standard labeled, specifying the NL option causes the tape to be treated as if it were nonlabeled. In this case, any file numbers specified in FILES statements must be physical file numbers, not logical file numbers.

  If you specify LABEL=SL in the PROC statement (for the output tape), do not specify NL on a subsequent INVOL statement. In other words, do not copy labeled and nonlabeled tapes onto the same output tape, unless the labeled tapes are to be treated as nonlabeled.

SL
: specifies that the input tape is standard labeled. Do not specify SL unless the tape is actually standard labeled.

BLP
: specifies that label processing is to be bypassed. BLP is the default if neither SL or NL is specified.

Be sure that you know the contents of any tape for which you specify BLP on an INVOL statement to avoid copying labeled and nonlabeled tapes to the same output tape.

DETACH
specifies that the tape drive be detached before issuing a mount command.

/'CMSmount'
specifies the tape mount command that your installation uses to mount a tape. Follow the slash with the text of the mount command enclosed in quotation marks. This option causes deferred mounting of the input tape; it must be used if you want deferred mounting. If this option is used, it must be specified as the last option on the INVOL statement.

At installations that do not have any mount commands, this option is invalid; therefore, you cannot use deferred mounting.

The mount request is executed via the standard CMS function call (SVC 202). If the return code is not zero (for example, if PROC TAPECOPY cannot find the specified mount command), the SAS System prints a message on the SAS log and the procedure stops processing. Some mount commands require that the tape drive be detached before the mount is issued (for example, VLIB). If this is a requirement, then DETACH must be specified on the INVOL statement.

FILES Statement

FILES *filenumbers*;
FILE *filenumbers*;

When you want to copy certain files from an input tape, use the FILES statement to specify the files you want to copy. Use as many FILES statements as you want. Depending on the kind of tape (labeled or nonlabeled) being copied and the intended label status of the output tape, you need to specify physical or logical file numbers in the FILES statement. **Table 50.1** shows what kind of file numbers to specify.

Table 50.1 Specifying File Numbers in the FILES Statement: PROC TAPECOPY

INPUT TAPE IS	LABEL= IS	OUTPUT TAPE IS	FILE # IS IN UNITS OF
SL	SL	SL	logical
SL	NL	NL	logical
NL	NL	NL	physical
NL	BLP	NL	physical
SL	BLP	SL	physical

Note: a physical file consists of the information on a tape between two tapemarks. A logical file consists of three physical files; the first containing a header

label; the second, data; and the third, a trailer label. Logical file implies a standard-labeled tape.

You can specify a range of files by putting a dash between the two files:

```
PROC TAPECOPY;
   FILES 1-7;
```

In a range, the second number must be greater than the first. The keyword EOV can be used as the second file in a range; PROC TAPECOPY copies all files on the input tape until the end of the volume (in most cases, a double tapemark). On a nonlabeled tape, you can copy files beyond the double tapemark by specifying the physical file number, counting tapemarks as usual. If another double tapemark exists on the input tape volume, you can then specify EOV in another range.

File numbers in a FILES statement can be specified in any order; you may want to copy file 5 and then file 2 and then file 1.

```
PROC TAPECOPY;
   FILES 5 2;
   FILE  1;
```

If you are using only one input tape, the FILES statement(s) can directly follow the PROC TAPECOPY statement. When you use several input tape volumes, follow each INVOL statement with the FILES statement(s).

DETAILS UNDER CMS

FILEDEF requirements FILEDEFs for input and output tapes are optional; however, if you use FILEDEFs, you need specify only the device on the FILEDEF command, for example, TAP*n*, where *n* is 1, 2, 3, or 4.

Printed Output

The TAPECOPY procedure prints a listing of the input and output tape characteristics and a summary of the files copied on the SAS log.

EXAMPLES

SL to SL under OS: Example 1

The job below copies a standard-labeled tape (volume serial XXXXXX) onto another standard-labeled tape (volume serial YYYYYY).

```
//jobname JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=tape,DISP=OLD,
//  VOL=SER=XXXXXX,LABEL=(,SL),DSN=first.dsname.on.tape
//VOLOUT DD UNIT=tape,DISP=(,KEEP),
//  VOL=SER=YYYYYY,LABEL=(,SL)
PROC TAPECOPY;
```

After PROC TAPECOPY executes, the output tape volume is labeled YYYYYY.

If LABEL=(,BLP) had been specified in the input tape DD statement (VOLIN), then it would not have been necessary to give the DSN= parameter. Since some installations do not permit the BLP label type specification and no volume label

checking is performed when it is specified, it is recommended that you specify (or allow to default) LABEL=(,SL).

The specification of LABEL=(,SL) in the output tape DD statement (VOLOUT) causes volume label checking to be performed by the operating system when a tape volume is mounted on the tape drive. The operating system ensures that a tape with volume serial YYYYYY is mounted. However, if the tape with external volume label YYYYYY were, in fact, internally labeled something other than YYYYYY, PROC TAPECOPY would fail. In this case, you would have to specify LABEL=(,BLP) or else give the actual internal volume serial in the output tape DD statement. If the output tape is not labeled internally, you can specify LABEL=(,NL) (as well as LABEL=(,BLP)).

SL to NL under OS: Example 2

The job below copies a standard-labeled tape with volume serial TAPEIN to a nonlabeled tape, FCSTP1. After executing the job, the output tape volume is still a nonlabeled tape, presumably with only an external volume label of FCSTP1. It is necessary to specify LABEL=NL in the PROC TAPECOPY statement; otherwise, the procedure defaults to LABEL=SL since the first (and only) input tape volume is standard labeled.

```
//jobname JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=tape,DISP=OLD,VOL=SER=TAPEIN,LABEL=(,BLP)
//VOLOUT DD UNIT=tape,DISP=(,KEEP),VOL=SER=FCSTP1,LABEL=(,NL)
PROC TAPECOPY LABEL=NL;
```

NL to NL under OS: Example 3

The job below copies a nonlabeled tape with volume serial QDR123 to a nonlabeled, 1600 bpi tape, SLXATK:

```
//jobname JOB account,name
//  EXEC SAS
//INTAPE DD UNIT=tape,DISP=OLD,VOL=SER=QDR123,LABEL=(,NL)
//OUTTAPE DD UNIT=2400-3,DISP=(,KEEP),VOL=SER=SLXATK,LABEL=(,NL)
PROC TAPECOPY INDD=INTAPE OUTDD=OUTTAPE DEN=1600;
```

Copying Several Files under OS: Example 4

The job below copies the first seven files from the standard-labeled input tape U02746 and four files from the standard-labeled input tape T13794 to an initially nonlabeled output tape with volume serial MINI01. The output tape is standard-labeled and has a volume serial of U02746 after the procedure is executed.

```
//jobname JOB account,name
//  EXEC SAS
//TAPI1 DD DISP=SHR,UNIT=tape,
//   VOL=SER=U02746,LABEL=(,SL),DSN=first.file.dsname
//TAPI2 DD UNIT=(tape,,DEFER)
//OUTDDN DD DISP=(,KEEP),UNIT=tape,VOL=SER=MINI01,LABEL=(,NL)
PROC TAPECOPY OUTDD=OUTDDN INDD=TAPI1 COPYVOLSER;
 FILES 3 2 1;
 INVOL INDD=TAPI2 INVOL=T13794
    DSN='first.dsname.on.this.tape';
 FILE 3;
```

```
INVOL INDD=TAPI1;
FILES 5-7 4;
INVOL INDD=TAPI2;
FILES 2 4 1;
```

Copying Multiple Files from Multiple Input Tapes under OS: Example 5

The job below copies several files from several input tape volumes onto one output tape volume:

```
//REARRNGE JOB account,name
//  EXEC SAS
//DEN2IN DD UNIT=(2400-4,,DEFER),LABEL=(,BLP)
//DEN3IN DD UNIT=(2400-3,,DEFER),LABEL=(,SL)
//TAPE1 DD UNIT=TAPE,DISP=SHR,VOL=SER=XR8475,LABEL=(,BLP)
//TAPE2 DD UNIT=TAPE,DISP=OLD,VOL=SER=BKT023,
// DSN=first.file.dsname
//OUTPUT DD UNIT=(3400-5,,DEFER),DISP=(,KEEP)
PROC TAPECOPY  LABEL=SL DEN=6250 NOLIST
  OUTDD=OUTPUT OUTVOL=HISTPE;
  INVOL INDD=DEN2IN INVOL=PTFTP0;
    FILES 2-4 8-EOV 7 6;
  INVOL INDD=TAPE1;
    FILES 5 7 9-EOV;
  INVOL INDD=TAPE2;
    FILES 4 5 1;
  INVOL INDD=TAPE1;
    FILE 1;
  INVOL INDD=DEN3IN INVOL=S03768 DSN='XRT.BKT120.G0081V00';
    FILES 1-6 22-34;
  INVOL INVOL=SO3760 DSN='T.BKT120.G0023V00';
    FILES 4 5 6 9;
  INVOL INDD=TAPE2;
    FILES 7-EOV;
```

Using Default Values under CMS: Example 6

In the example below, PROC TAPECOPY defaults to TAP1 for input and TAP2 for output. The output tape volume is labeled with the same label as the input tape. All files, including labels, are copied because LABEL=BLP is assumed.

```
PROC TAPECOPY;
```

SL to NL under CMS: Example 7

The job below copies a standard-labeled tape; after executing PROC TAPECOPY, the output tape volume is a nonlabeled tape. It is necessary to specify LABEL=NL in the PROC TAPECOPY statement because TAPECOPY would otherwise default to LABEL=BLP and copy all files (both data and labels).

```
PROC TAPECOPY LABEL=NL NOLIST;
```

Using FILEDEFs with PROC TAPECOPY: Example 8

The job below copies a tape to a 1600bpi tape. BLP is assumed for the label status of both tapes.

```
CMS FILEDEF INTAPE TAP1;
CMS FILEDEF OUTTAPE TAP2;
PROC TAPECOPY INDD=INTAPE OUTDD=OUTTAPE DEN=1600 NOLIST;
```

Copying Multiple Files from Multiple Tapes: Example 9

The job below copies files from four standard-labeled input tapes to one output tape (which becomes standard labeled). Since there is no LABEL= specification in the PROC statement, it defaults to BLP. An INVOL statement is used for each input tape to specify that the tapes are SL and to provide the deferred tape mount command. Each tape must be standard labeled, or PROC TAPECOPY fails.

```
PROC TAPECOPY TAP1 NOLIST TAP2 COPYVOLSER;
   INVOL SL /'MOUNT T13794 ON 181 NORING';
   FILE 3;
   INVOL SL /'MOUNT TXXXXX ON 181 NORING';
   FILES 4 5-7;
   INVOL SL /'MOUNT TYYYYY ON 181 NORING';
   FILES 2 4 1 9-EOV;
```

Chapter 51

The TAPELABEL Procedure

Operating systems: CMS and OS

ABSTRACT

The TAPELABEL procedure lists the label information of an IBM standard-labeled tape volume under OS batch, TSO, and CMS.

INTRODUCTION

One or more standard-labeled tape volumes can be processed by TAPELABEL. Only one volume per job control statement or command is processed; however, multiple job control statements or commands can be used in one job to process more than one tape volume. At some installations, it may be necessary to specify the data set name of the first file on the tape volume in the job control describing the volumes to be processed.[1]

The procedure prints information from the tape label, including the data set name, DCB information, and data set history.

SPECIFICATIONS

The only statement used with this procedure is the PROC statement.

PROC TAPELABEL *options*;

PROC TAPELABEL Statement

PROC TAPELABEL *options*;

The options below can appear in the PROC TAPELABEL statement:

DDNAME= (*fileref* ...) specifies the *fileref* assigned to the tape volume to be processed. More than one fileref can be specified. If you specify only one fileref, you can omit the parentheses. [2]
Operating systems: CMS and OS

MAP gives for every file on the tape:

- the number of blocks
- the length in bytes of the largest and the smallest block in the file
- the density with which the tape was written
- the length in feet.

Operating system: CMS

PAGE begins the output for each tape volume on a new page.
Operating systems: CMS and OS

TAP*n* specifies a CMS tape symbolic address, where *n* can be a value of 1, 2, 3, or 4. TAP1 corresponds to virtual address 181, TAP2 to 182, TAP3 to 183, and TAP4 to 184. This option can be used instead of the DDNAME= option to specify a tape volume to be processed. A corresponding FILEDEF command is unnecessary if TAP*n* is used.
Operating system: CMS

DETAILS

Printed Output

For each file on a tape volume, TAPELABEL prints:

1. FILE NUMBER, the file sequence number
2. DSNAME, the data set name
3. RECFM, the record format
4. LRECL, the logical record length
5. BLKSIZE, the block size
6. BLOCK COUNT, the number of blocks in the file (from trailer label)
7. EST. FEET, the estimated length of the file in feet
8. CREATED, the file creation date
9. EXPIRES, the file expiration date
10. CREATED BY JOB NAME STEPNAME, the job and step names of the job that created the file
11. TRTCH, the track recording technique
12. DEN, the file recording density code
13. PSWD, the file protection indicator
14. UHL, the number of user header labels
15. UTL, the number of user trailer labels

TAPELABEL also prints the sum of the estimated file lengths.

EXAMPLES

OS Example: Example 1

The job below lists the label information for all files on tape volumes referenced by the filerefs TAPE1 and TAPE2. The UNIT=AFF=TAPE1 specification in the TAPE2 DD statement indicates that the same tape drive is to be used for both TAPE1 and TAPE2.

```
//jobname JOB acct,name
//  EXEC SAS
//TAPE1 DD UNIT=TAPE,VOL=SER=USG824,DISP=OLD
//TAPE2 DD UNIT=AFF=TAPE1,VOL=SER=GRP824,DISP=OLD
PROC TAPELABEL DDNAME=(TAPE1 TAPE2);
   TITLE 'LABEL INFO FOR FILES ON TAPE1 AND TAPE2 VOLUME FILES TAPE
      LIST FOR DDNAME - TAPE 1';
```

Output 51.1 Label Information for Files on Tape1 and Tape2 Volumes

LABEL INFO FOR TAPE1 AND TAPE2 TAPE VOLUME FILES

TAPE LIST FOR DDNAME - TAPE1

CONTENTS OF TAPE VOLUME - TAPE24 OWNER -

FILE NUMBER	DSNAME	RECFM	LRECL	BLKSIZE	BLOCK COUNT	EST. FEET	CREATED	EXPIRES	CREATED BY JOB NAME	CREATED BY STEPNAME	TRTCH	DEN	PSWD	UHL	UTL
1	SAS.TEXT	FBA	80	32720	1	1.5	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
2	SAS.CNTL	FB	80	32720	2	1.9	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
3	SAS.UTEXT	VBA	137	32720	3	2.4	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
4	SAS.UCNTL	FB	80	32720	2	1.9	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
5	SAS.USAGE	U	0	32760	119	56.0	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
6	SAS.ZAP	FB	80	8000	65	9.6	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
7	SAS.UHELP	FB	80	8000	1	1.1	20SEP84	0000000	UTAPE	/SAS		4	NO	0	0
						74.7									

TAPE LIST FOR DDNAME - TAPE2

CONTENTS OF TAPE VOLUME - TAPE34 OWNER - SAS

FILE NUMBER	DSNAME	RECFM	LRECL	BLKSIZE	BLOCK COUNT	EST. FEET	CREATED	EXPIRES	CREATED BY JOB NAME	CREATED BY STEPNAME	TRTCH	DEN	PSWD	UHL	UTL
1	SAS.TEXT	FBA	80	32720	1	1.5	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
2	SAS.CNTL	FB	80	32720	2	1.9	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
3	SAS.GTEXT	VBA	137	32720	10	5.6	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
4	SAS.GCNTL	FB	80	32720	3	2.4	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
5	SAS.GLOAD	VB	32756	32760	136	63.8	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
6	SAS.GMAPS	U	32756	32760	185	86.4	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
7	SAS.GSOURCE	FB	80	32720	3	2.4	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
8	SAS.GREPLAY	U	32756	32760	112	52.7	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
9	SAS.GSAMPLE	FB	80	32720	5	3.3	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
10	SAS.GMACLIB	FB	80	32720	1	1.5	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
11	SAS.GHELP	FB	80	8000	2	1.3	06OCT83	0000000	GTAPE	/SAS		4	NO	0	0
						223.4									

OS Example: Example 2

The following is an example of the LABEL=(,BLP) specification, which indicates that you want to bypass label processing.

```
//jobname JOB acct,name
//  EXEC SAS
//XYZ DD UNIT=2600,VOL=SER=123456,DISP=OLD,LABEL=(,BLP)
PROC TAPELABEL DDNAME=XYZ;
```

There is no output for this example.

CMS Example: Example 3

The statements below list the label information for all files on the tape volume currently mounted on virtual address 182.

```
PROC TAPELABEL TAP2;
   TITLE 'LABEL INFO FOR VA182 TAPE VOLUME FILES';
   RUN;
```

Output 51.2 Label Information for VA182 Tape Volume Files

```
 TAPE LIST FOR DD/DEVICE-  181
CONTENTS OF TAPE VOLUME  -  TAP796
                                   OWNER -  SAS INST

FILE                                                BLOCK   EST.
NUMBER DSNAME            RECFM  LRECL  BLKSIZE     COUNT   FEET    CREATED EXPIRE


   1   SAS.INSTRUCT        VBA     137   32720         7     4.5   31MAR82 000000
   2   SAS.CNTL            FB       80   32720         5     3.5   26MAR82 000000
   3   SAS.OBJECT          FB       80     800        28     2.8   25MAR82 000000
   4   SAS.LIBRARY         VB    32756   32760        32    16.7   25MAR82 000000
   5   SAS.LIBRARYO        VB    32756   32760       148    73.1   31MAR82 000000
   6   SAS.LIBRARYF        VB    32756   32760       177    87.3   31MAR82 000000
   7   SAS.SLIBRARY        VB    32756   32760        35    18.1   25MAR82 000000
   8   SAS.SLIBRAYO        VB    32756   32760        19    10.3   25MAR82 000000
   9   SAS.SLIBRAYF        VB    32756   32760        34    17.6   25MAR82 000000
  10   SAS.PLFTRAN         VB    32756   32760         3     2.6   25MAR82 000000
  11   SAS.OUTPUT          VBA     137   32720         7     4.5   25MAR82 000000
  12   SAS.SAMPLE          FB       80   32720        40    20.5   25MAR82 000000
  13   SAS.SUBLIB          VB    32756   32760         3     2.6   25MAR82 000000
  14   SAS.SUBLIBO         VB    32756   32760         7     4.5   25MAR82 000000
  15   SAS.SUBLIBF         VB    32756   32760         8     5.0   25MAR82 000000
                                                                                2
  16   SAS.MACLIB          FB       80   32720        17     9.4   25MAR82 000000
  17   SAS.MACLIBP         FB       80   32720         4     3.0   25MAR82 000000
  18   SAS.SOURCE          FB       80   32720        29    15.2   25MAR82 000000
  19   SAS.SSOURCE         FB       80   32720       105    52.1   25MAR82 000000
  20   SAS.MISCLIB         VB    32756   32760         4     3.0   25MAR82 000000
                                                                   ------
                                                                   357.4
```

REFERENCE

International Business Machines Corporation (1983), *MVS/XA Tape Labels*, Order Number GC26-4003.

NOTES

1. **OS**: This applies only if you cannot use LABEL=(,BLP).
2. **OS**: If DDNAME= is omitted, the default fileref is TAPE.

CMS: If DDNAME= is omitted, the default fileref is TAP1.

Chapter 52

The TIMEPLOT Procedure

Operating systems: All

ABSTRACT

The TIMEPLOT procedure is used to plot one or more variables over time intervals.

INTRODUCTION

For each plot requested, PROC TIMEPLOT produces a plot and a listing of observations in the data set similar to that produced by PROC PLOT and PROC PRINT.

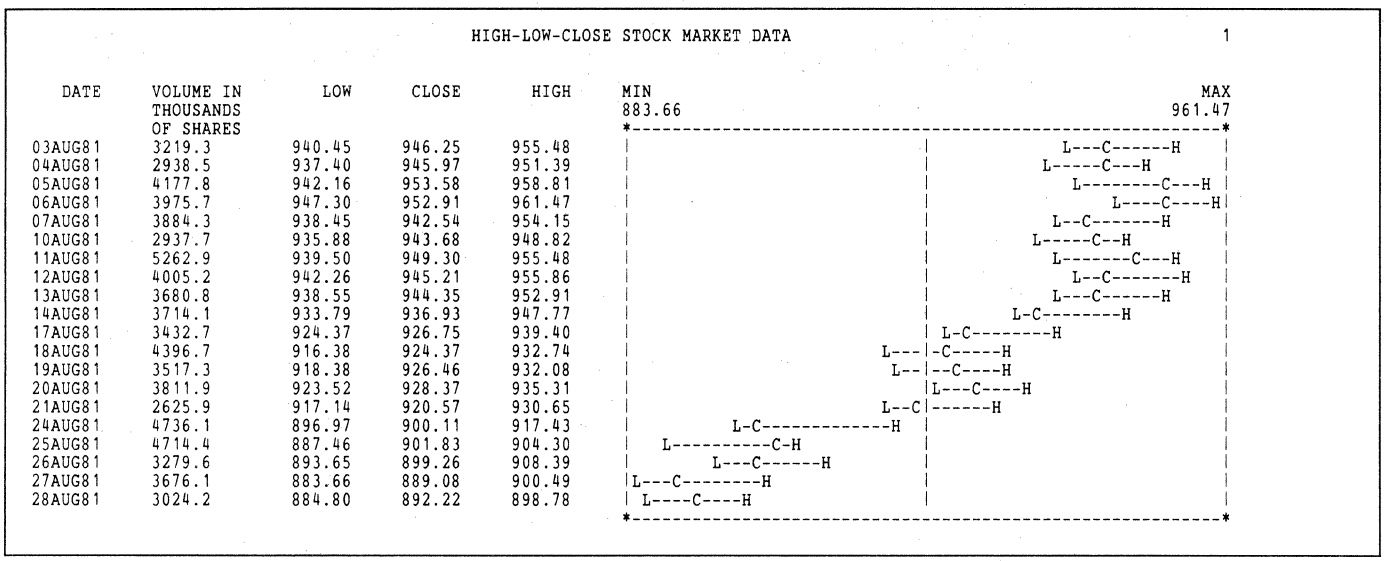
However, the TIMEPLOT procedure has several distinctive features:

- The vertical axis always represents the sequence of observations in the data set; thus, if the observations are arranged in date or time order, the vertical axis represents time.
- The horizontal axis contains values of the variable or variables you are examining.
- A plot produced by TIMEPLOT may occupy more than one page.
- Each observation appears sequentially on a separate line of the plot; no observations are "hidden" as may occur with the PLOT procedure.
- Each observation in the plot is accompanied by a print of the values plotted.

PROC TIMEPLOT is especially useful for combining multiple observations into a single line in the plot. TIMEPLOT can also overlay multiple plots on a single set of axes, as PLOT does.

The following example illustrates one type of plot produced by TIMEPLOT. In this example, the plotting section contains the high, low, and closing values of the Dow Jones Index of prices on the New York Stock Exchange. The symbols for the variables are joined by a line. The minimum and maximum values label the horizontal axis; a vertical reference line appears at the mean low value. The listing on the left side of the output contains, for each observation, the date and volume of stocks traded; and the low, closing, and high prices for that day are printed for each observation. The statements that produced this example appear in the **Examples** section at the end of the chapter.

Output 52.1 High-Low-Close Stock Market Report: PROC TIMEPLOT



HIGH-LOW-CLOSE STOCK MARKET DATA 1

DATE	VOLUME IN THOUSANDS OF SHARES	LOW	CLOSE	HIGH
03AUG81	3219.3	940.45	946.25	955.48
04AUG81	2938.5	937.40	945.97	951.39
05AUG81	4177.8	942.16	953.58	958.81
06AUG81	3975.7	947.30	952.91	961.47
07AUG81	3884.3	938.45	942.54	954.15
10AUG81	2937.7	935.88	943.68	948.82
11AUG81	5262.9	939.50	949.30	955.48
12AUG81	4005.2	942.26	945.21	955.86
13AUG81	3680.8	938.55	944.35	952.91
14AUG81	3714.1	933.79	936.93	947.77
17AUG81	3432.7	924.37	926.75	939.40
18AUG81	4396.7	916.38	924.37	932.74
19AUG81	3517.3	918.38	926.46	932.08
20AUG81	3811.9	923.52	928.37	935.31
21AUG81	2625.9	917.14	920.57	930.65
24AUG81	4736.1	896.97	900.11	917.43
25AUG81	4714.4	887.46	901.83	904.30
26AUG81	3279.6	893.65	899.26	908.39
27AUG81	3676.1	883.66	889.08	900.49
28AUG81	3024.2	884.80	892.22	898.78

MIN 883.66 MAX 961.47

SPECIFICATIONS

The following statements are used to control the TIMEPLOT procedure:

PROC TIMEPLOT *options*;
PLOT *requests / options*;
CLASS *variables*;
BY *variables*;
ID *variables*;

No more than one CLASS, ID, and BY statement should appear. There is no restriction on the number of PLOT statements that can be given.

PROC TIMEPLOT Statement

PROC TIMEPLOT *options*;

The following options are used with the PROC TIMEPLOT statement:

UNIFORM indicates that the scale used for a given plot request is to be the same for all BY groups. If UNIFORM is omitted and a BY statement is used, the scale is determined for each BY group separately. In addition, UNIFORM indicates that a REF= specification (see

below) of the form MEAN(*variables*) is to represent the mean over all observations for all BY groups.

MAXDEC=*n* indicates the maximum number of decimal positions printed for a number. A decimal specification in a format overrides a MAXDEC= specification. The default is MAXDEC=2.

PLOT Statement

PLOT *specifications / options;*

The PLOT statement specifies the plots to be produced. One PROC TIMEPLOT statement can be followed by several PLOT statements, and each PLOT statement can request several plots.

Each plot specification consists of a variable with an optional symbol variable or value. Unless the OVERLAY option is used, each variable specified appears in a separate plot. A plot specification can be given as any of the following types:

variable1...variablen
variable1=symbol1...variablen=symboln
(variables)=symbol1...(variables)=symboln

Variable1...variablen indicates that each variable is to be plotted using the default symbol, the first character of the variable name. For example, the statement

```
PLOT GROSS NET;
```

causes the values of GROSS to be plotted with a G and the values of NET with an N.

Variable1=symbol1... variablen=symboln indicates that each variable is to be plotted using the symbol specified. The symbol may be a single character enclosed in quotes or a variable name. If a variable name is given, the first nonblank character of the formatted value of the variable is used as the plotting symbol. The symbol variable may be either character or numeric. In this example:

```
PLOT GROSS='*' NET=PROFLOSS;
```

the values of GROSS are plotted using an asterisk (*) and the values of NET are plotted using the first nonblank character of the value of PROFLOSS for that observation.

(Variables)=symbol1...(variables)=symboln indicates that each variable within the parentheses is to be plotted using the symbol associated with that group. The symbol may be a single character enclosed in single quotes or a variable name. If a variable name is given, the first nonblank character of the formatted value of the variable is used as the plotting symbol. For example, the statement

```
PLOT (NC SC VA GL FL)='*'
     (AL AR MS LA TX)='+';
```

plots the values of the variables in the first group using an asterisk (*) and the values of variables in the second group using a cross (+).

A single PLOT statement can contain requests in more than one of these forms.

The listing on the left side of the output contains different information depending on whether a CLASS statement is used. If no CLASS statement is used, each observation is printed (and plotted) on a separate line. If a CLASS statement is used without a symbol variable being specified, all observations for a given combination of class variable values are plotted on the same line; the last value of the plotting variable appears in the print column, thus omitting some values from

printing and plotting. If a CLASS statement and a symbol variable are used, all observations for a given combination of CLASS variable values are plotted on the same line, using the first-character values of the symbol variable as plotting characters. In the print section, each value of the symbol variable has a different column, and the value of the plotting variable is printed in the column for its symbol value. If more than one observation within a class has the same value of a symbol variable, only the first occurrence of the value is plotted and printed, and a warning message appears on the log.

The following options are used with the PLOT statement:

OVERLAY
: specifies that all the plot specifications requested in the PLOT statement are to be printed in the same plot. Otherwise, each specification is printed separately.

HILOC
: indicates that the leftmost non-reference symbol will be connected to the rightmost non-reference symbol by a line of hyphens (-). The HILOC option is ignored if the JOINREF option is also specified. This option is often used in HIGH-LOW-CLOSE applications.

JOINREF
: indicates that the leftmost symbol is to connect to the rightmost symbol, regardless of whether the symbols are reference symbols or plotting symbols. However, if a line contains only reference symbols, the symbols are not connected.

REVERSE
: indicates that the plot is to be printed with the values of the horizontal axis descending, so that the maximum value is plotted in the leftmost plot position and the minimum value is plotted in the rightmost position. This order of plotting is also used if the AXIS= values are in descending order. Therefore, if AXIS= is specified with descending values, the REVERSE option is ignored.

POS=*n*
: indicates the number of positions comprising the plotting section. If you specify POS=0 and also specify the AXIS= option, the plot is expanded to fill the page, with each axis value comprising several plot positions instead of just one. If POS= and AXIS= are both omitted, an initial POS= value of 20 is assumed, but it may be increased to fill the page width along with the value section. See **Details**, below, for more information.

AXIS=*specification*
: indicates the tick marks for the horizontal axis. When you specify an AXIS= value, each tick mark of the axis range consists of one plot column unless the POS= option is also specified. (See the POS= option for more information.) Examples of axis specifications include:

Specification	Tick marks
AXIS=10 TO 100 BY 10	10, 20, 30...100
AXIS=10,30,40	10, 30, 40
AXIS=10,20,30 TO 40 BY 2	10, 20, 30, 32...40
AXIS='01JAN85'D TO '01JAN86'D BY MONTH	01JAN85 01FEB85... 01JAN86

AXIS='01JAN85'D TO '01JAN86'D BY QTR	01JAN85 01APR85... 01JAN86

In the last two examples, the FROM and TO values can be any of the valid SAS date, time, or datetime values described for the SAS functions INTCK and INTNX (see the chapter "SAS Functions"). The BY value can be any of the valid values listed for the *interval* argument in the SAS functions INTCK and INTNX. You must use a FORMAT statement to print the tick mark values in an understandable form.

If an AXIS= list is given, the scale of the plot is not modified to encompass all values of plotting variables or reference lines. Any plotting value outside the range specified by AXIS= is indicated by a < or > on the left or right border of the plot, respectively. A REF= value outside the AXIS= range is ignored.

REF=*values*
indicates the position of one or more vertical reference lines in the plotting section. Constant values may be given, as well as values in the form

MEAN(*variables*)

If values of this form are given, the mean is evaluated for each variable in parentheses, and a reference line is drawn for each.

If UNIFORM is specified in the TIMEPLOT statement, the MEAN values are the calculated mean for the variables over all observations for all BY groups.

If any REF= value is less than the minimum value of any plotting variable, or if any REF= value is greater than the maximum of any plotting variable, the scale of the plot is adjusted to allow the reference line to be printed (unless the AXIS= option is used). Reference lines are printed using the character specified in the REFCHAR= option (or |, if none is specified). If a plotting variable value is to be plotted in the column where a REF= value goes, the plotting symbol is printed instead of the REFCHAR= value.

REFCHAR=*charactervalue*
indicates the character to print for reference lines. If no REFCHAR= option is given, the vertical bar (|) is used. If you are using the JOINREF or HILOC option, you should not use a REFCHAR that is the same as any plotting symbol since TIMEPLOT does not connect REFCHAR symbols to each other.

OVPCHAR=*charactervalue*
indicates the character to be printed if two plotting symbols are to be plotted in the same column. If no OVPCHAR= option is given, @ is used. Note that if a plotting symbol is to be plotted on the same column as a REFCHAR value, the plotting symbol is printed, not the OVPCHAR value. TIMEPLOT never actually overprints two symbols.

NOSYMNAME
controls whether symbol variable names are printed in the column headings for symbol values when a CLASS statement is used. If the NOSYMNAME option is used, only the value of the symbol variable appears in the column heading, as in

VALUE
PLOTTING VARIABLE

If NOSYMNAME is omitted, headings have the form

SYMBOLNAME: VALUE
PLOTTING VARIABLE

If a CLASS statement is not used, this option is ignored.

CLASS Statement

CLASS *variables;*

If a CLASS statement is used, only one line is printed for each combination of class variable values. It is not necessary that the data be sorted by the class variables, but the output may be more meaningful if the data are grouped according to values of the class variables. (The grouping of data is similar to applications of other procedures that use the NOTSORTED option in a BY statement.) Class variables can be numeric or character. The CLASS statement is normally used only in conjunction with a symbol variable. Class variable values are printed, but not plotted. Only one CLASS statement can be used.

BY Statement

BY *variables;*

A BY statement can be used with PROC TIMEPLOT to obtain separate plots for observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step." The BY statement should appear only once.

ID Statement

ID *variables;*

The ID statement is used to print variables in the printed value section without plotting them. ID variables can be numeric or character. Only one ID statement can be used.

DETAILS

Missing Values

A missing value of a plotting variable appears as a missing value (. or one of the special missing values A-Z and underscore) in the listing, and it is not plotted.

If a CLASS statement and a symbol value are used, the observations in a given class may not contain all values of the symbol variable. For example, if the values of the symbol variable are 'PREDICTED' and 'ACTUAL', classes representing future dates may not have a value for 'ACTUAL'. In this case the value of ACTUAL is represented in the print section by a blank and is not plotted.

A missing value of a CLASS variable is treated as a valid level of the variable and defines a class of observations. The missing value appears as a missing value in the print section. A missing value of an ID variable is printed as a missing value.

A missing value of a symbol variable is treated like any other value of that variable. A print column is established for that value and the value is plotted. However, even if the missing value is a blank (as occurs when the symbol variable is a character variable), the column heading and the plot represent the value as a period (.) .

Input Data

The input data set usually contains a date variable to be used as either a CLASS or an ID variable. Although the data set does not have to be sorted by date, the output is usually more meaningful if the data are in chronological order. In addition, if a CLASS statement is used, the output is more meaningful if observations are grouped according to combinations of CLASS variable values. CLASS, ID, BY, and symbol variables can be numeric or character; plotting variables must be numeric.

Printed Output

The procedure prints one or more pages of printed values and plots for each plot request. The page is divided into two sections: the listing and the plot.

The listing contains printed values for:

1. CLASS variables.
2. ID variables (the value printed for each ID variable is the first value in that class group).
3. plotting variables.
4. column headings for CLASS and ID variables. These consist of either variable labels (if they are present and if space permits) or variable names. To suppress variable labels in column headings, use the NOLABEL system option. (See the OPTIONS statement in "SAS Statements Used Anywhere" for a description of the NOLABEL option).
5. column headings for plotting variables. These consist of two parts: the variable label or name, and the name (unless NOSYMNAME is used) and value of the symbol variable (if a CLASS statement and a symbol variable are both used). If a format is associated with a variable, the value is printed using that format.

The plot contains:

6. plotted values of variables.
7. tick marks.
8. reference lines. The maximum and minimum values of the horizontal axis are the maximum and minimum values of the data. The plot is labeled with the minimum and maximum values being plotted, if space permits.
9. If a POS= value is given, the page is arranged as follows: (1) the size of the plot is determined from the POS= values; (2) the space for the listing is determined from the width of the columns of printed values, equally spaced and with a maximum of five positions between columns; (3) the output is centered on the page. If a POS= value is not given, the width of the listing is determined, and the plot is expanded to fill the rest of the page.
10. If there is not sufficient room for all values to be printed along with a plot, an error message is printed on the log, and that plot is not printed. Other plots are not affected, however.

EXAMPLES

How Well a Model Fits the Data: Example 1

This example uses TIMEPLOT to demonstrate how well a model fits actual data. The REG procedure is used to produce the data. First, TIMEPLOT overlays the plot of a predicted and an actual value. The mean of the actual values is given as a reference line. Next, TIMEPLOT shows how residuals digress from the origin by using the REF=0 option to produce a reference line at the origin and the JOINREF option to connect the residuals with the origin.

```
DATA GRUNFELD;
   INPUT YEAR I F C @@;
   LABEL I='GROSS INVESTMENT GE'
         C='CAPITAL STOCK LAGGED GE'
         F='VALUE OF SHARES GE LAGGED';
   CARDS;
1935  33.1  1170.6  97.8   1936  45.0  2015.8 104.4
1937  77.2  2803.3 118.0   1938  44.6  2039.7 156.2
1939  48.1  2256.2 172.6   1940  74.4  2132.2 186.6
1941 113.0  1834.1 220.9   1942  91.9  1588.0 287.8
1943  61.3  1749.4 319.9   1944  56.8  1687.2 321.3
1945  93.6  2007.7 319.6   1946 159.9  2208.3 346.0
1947 147.2  1656.7 456.4   1948 146.3  1604.4 543.4
1949  98.3  1431.8 618.3   1950  93.5  1610.5 647.4
1951 135.2  1819.4 671.3   1952 157.3  2079.7 726.1
1953 179.5  2371.6 800.3   1954 189.6  2759.9 888.9
;
PROC REG;
   MODEL I = F C ;
   OUTPUT OUT=C R=RESID P=PRED;
   TITLE 'GRUNFELD''S INVESTMENT MODEL';
PROC TIMEPLOT DATA=C;
   PLOT PRED='P' I='A'/OVERLAY REF=MEAN(I);
   PLOT RESID/JOINREF REF=0;
```

Output 52.2 Fitting a Model to Actual Data: PROC TIMEPLOT

```
                                GRUNFELD'S INVESTMENT MODEL                    1

DEP VARIABLE: I        GROSS INVESTMENT GE
                                ANALYSIS OF VARIANCE

                                  SUM OF          MEAN
                SOURCE     DF     SQUARES        SQUARE      F VALUE      PROB>F

                MODEL       2  31632.03023  15816.01511       20.344      0.0001
                ERROR      17  13216.58777    777.44634
                C TOTAL    19  44848.61800

                    ROOT MSE      27.88272     R-SQUARE       0.7053
                    DEP MEAN        102.29     ADJ R-SQ       0.6706
                    C.V.           27.2585
```

(continued on next page)

(continued from previous page)

VARIABLE	DF	PARAMETER ESTIMATE	STANDARD ERROR	T FOR H0: PARAMETER=0	PROB > |T|	VARIABLE LABEL
INTERCEP	1	-9.95630645	31.37424914	-0.317	0.7548	INTERCEPT
F	1	0.02655119	0.01556610	1.706	0.1063	VALUE OF SHARES GE LAGGED
C	1	0.15169387	0.02570408	5.902	0.0001	CAPITAL STOCK LAGGED GE

GRUNFELD'S INVESTMENT MODEL 2

PREDICTED VALUE	GROSS INVESTMENT GE	MIN 33.1 ... MAX 198.163
35.96	33.10	AP
59.40	45.00	A P
82.37	77.20	A P
67.89	44.60	A P
76.13	48.10	A P
74.96	74.40	a
72.25	113.00	P A
75.86	91.90	P A
85.02	61.30	A P
83.58	56.80	A P
91.83	93.60	PA
101.16	159.90	P A
103.26	147.20	P A
115.07	146.30	P A
121.85	98.30	A P
131.01	93.50	A P
140.18	135.20	A P
155.41	157.30	PA
174.41	179.50	P A
198.16	189.60	A P

GRUNFELD'S INVESTMENT MODEL 3

RESIDUALS	MIN -37.511 ... MAX 58.73724
-2.86	R---
-14.40	R---
-5.17	R---
-23.29	R---
-28.03	R---
-0.56	R
40.75	---R
16.04	---R
-23.72	R---
-26.78	R---
1.77	R
58.74	---R
43.94	---R
31.23	---R
-23.55	R---
-37.51	R---
-4.98	R---
1.89	-R
5.09	----R
-8.56	R---

Interrelationship of Forecast and Actual Values: Example 2

This example plots forecasting data. The FORECAST procedure is run on time series data from two industries, and an actual and a forecast value are output for each industry. TIMEPLOT produces a plot for each industry. For each plot, TIMEPLOT plots the forecast and actual values on the same line with the JOINREF option to show how they interrelate. The NOSYMNAME option prevents the variable name _TYPE_ from being printed. The third plot shows the use of the AXIS= option. The range is specified as 12-77 (by 5), with any values out of that range

indicated on the plot border. The fourth plot shows the use of the AXIS= option with the POS= option.

```
DATA D1;
   INPUT TRANS @@;
   LABEL TRANS='TRANSPORTATION EQUIPMENT';
   CARDS;
14 16 13 17  14 14 13 17  10 12 11 12  08 13 16 20  18 26 28 28
23 25 24 26  21 23 25 36  28 36 34 34  24 29 33 40  37 47 52 53
38 42 34 31  22 22 21 20  18 22 27 28  25 32 34 34  22 28 26 34
26 33 36 38  31 40 41 46  37 49 52 61  52 64 69 70
;
DATA D2;
   INPUT STONE @@;
   LABEL STONE='STONE CLAY GLASS';
   CARDS;
06 09 09 08  06 07 06 08  04 04 03 05  04 07 08 11  08 12 12 14
10 09 09 10  09 10 10 11  09 11 10 15  11 13 16 24  17 22 23 26
18 20 18 19  17 15 11 12  13 17 21 18  18 20 18 19  14 16 18 22
14 19 17 20  16 17 17 20  17 18 17 22  19 23 23 27
;
DATA C;
   MERGE D1 D2;
   QTR=MOD(_N_-1,4)+1;
   YEAR=INT((_N_-1)/4)+1947;
   DATE=YYQ(YEAR,QTR);
   FORMAT DATE YYQ4.;
PROC FORECAST DATA=C OUT=O1 OUTEST=OE AR=5
   OUTDATA OUT1STEP OUTLIMIT
   INTERVAL=QTR;
   ID DATE;
   VAR TRANS STONE;
PROC TIMEPLOT DATA=O1 UNIFORM;
   PLOT (TRANS STONE)=_TYPE_/JOINREF NOSYMNAME;
   CLASS DATE;
   TITLE 'FORECASTING EXAMPLE';
DATA TRANS;
   SET O1;
   IF _TYPE_='ACTUAL' AND TRANS¬=.;
PROC TIMEPLOT;
   PLOT TRANS/AXIS=12 TO 77 BY 5;
   PLOT TRANS/AXIS=12 TO 77 BY 5 POS=0;
```

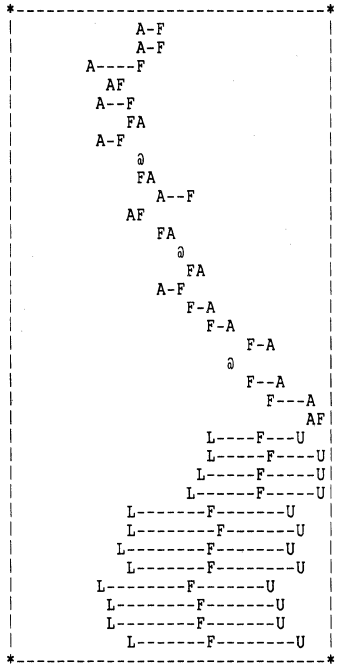
Output 52.3 Plotting Actual and Forecast Data: PROC TIMEPLOT

FORECASTING EXAMPLE 1

DATE	❶ ACTUAL TRANSPORTATION EQUIPMENT	❷ FORECAST TRANSPORTATION EQUIPMENT	L95 TRANSPORTATION EQUIPMENT	U95 TRANSPORTATION EQUIPMENT	MIN 8 MAX 73.65622
47Q1	14.00	12.50			FA
47Q2	16.00	14.28			FA
47Q3	13.00	16.09			A-F
47Q4	17.00	13.18			F-A
48Q1	14.00	17.41			AF
48Q2	14.00	15.79			AF
48Q3	13.00	11.77			@
48Q4	17.00	16.20			@
49Q1	10.00	15.42			A--F
49Q2	12.00	11.41			@
49Q3	11.00	11.69			AF
49Q4	12.00	15.84			A-F
50Q1	8.00	9.26			AF
50Q2	13.00	11.59			@
50Q3	16.00	14.01			FA
50Q4	20.00	18.94			FA
51Q1	18.00	18.11			@
51Q2	26.00	21.56			F-A
51Q3	28.00	26.88			@
51Q4	28.00	30.16			AF
52Q1	23.00	24.73			AF
52Q2	25.00	26.64			AF
52Q3	24.00	25.16			@
52Q4	26.00	24.37			@
53Q1	21.00	22.38			AF
53Q2	23.00	22.66			@
53Q3	25.00	22.21			FA
53Q4	36.00	27.18			F---A
54Q1	28.00	31.90			A-F
54Q2	36.00	28.87			F--A
54Q3	34.00	33.67			@
54Q4	34.00	40.18			A--F
55Q1	24.00	26.54			AF
55Q2	29.00	28.52			@
55Q3	33.00	26.94			F--A
55Q4	40.00	34.03			F--A
56Q1	37.00	32.33			F--A
56Q2	47.00	38.22			F---A
56Q3	52.00	45.28			F--A
56Q4	53.00	52.88			@
57Q1	38.00	45.42			A---F
57Q2	42.00	39.83			FA
57Q3	34.00	40.74			A--F
57Q4	31.00	35.00			A-F
58Q1	22.00	20.77			FA
58Q2	22.00	26.40			A-F
58Q3	21.00	19.40			FA
58Q4	20.00	23.38			AF
59Q1	18.00	18.26			@
59Q2	22.00	22.17			@
59Q3	27.00	25.34			FA
59Q4	28.00	30.21			AF
60Q1	25.00	29.52			A-F
60Q2	32.00	29.78			FA

FORECASTING EXAMPLE 2

DATE	ACTUAL TRANSPORTATION EQUIPMENT	FORECAST TRANSPORTATION EQUIPMENT	L95 TRANSPORTATION EQUIPMENT	U95 TRANSPORTATION EQUIPMENT
60Q3	34.00	36.85		
60Q4	34.00	37.14		
61Q1	22.00	33.03		
61Q2	28.00	28.72		
61Q3	26.00	31.33		
61Q4	34.00	31.26		
62Q1	26.00	28.85		
62Q2	33.00	33.58		
62Q3	36.00	33.11		
62Q4	38.00	44.31		
63Q1	31.00	34.42		
63Q2	40.00	37.21		
63Q3	41.00	42.68		
63Q4	46.00	44.86		
64Q1	37.00	41.37		
64Q2	49.00	43.63		
64Q3	52.00	48.50		
64Q4	61.00	56.12		
65Q1	52.00	52.31		
65Q2	64.00	57.34		
65Q3	69.00	60.89		
65Q4	70.00	71.81		
66Q1		58.63	49.30	67.96
66Q2		61.12	48.85	73.40
66Q3		59.53	45.40	73.66
66Q4		58.58	43.87	73.28
67Q1		48.96	32.25	65.67
67Q2		49.43	31.86	66.99
67Q3		47.61	29.62	65.61
67Q4		48.63	30.63	66.63
68Q1		43.55	25.44	61.67
68Q2		45.63	27.51	63.75
68Q3		45.82	27.68	63.97
68Q4		49.04	30.61	67.47



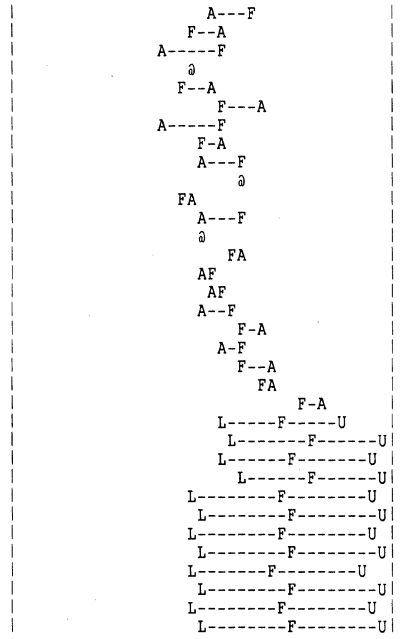
FORECASTING EXAMPLE 3

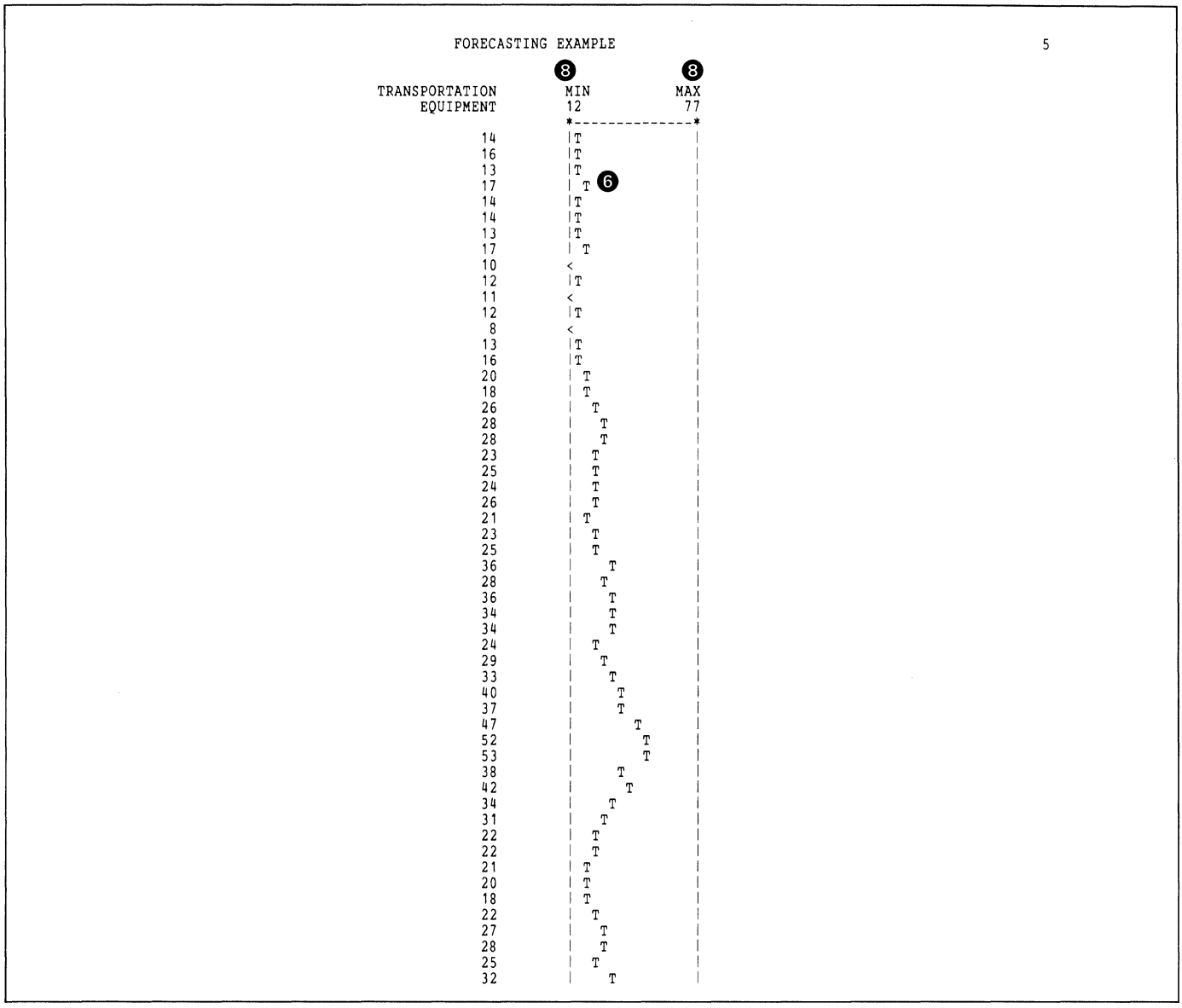
❷

DATE	ACTUAL STONE CLAY GLASS	FORECAST STONE CLAY GLASS	L95 STONE CLAY GLASS	U95 STONE CLAY GLASS	MIN 3 / MAX 32.11593
47Q1	6.00	6.16			
47Q2	9.00	6.27			
47Q3	9.00	8.28			
47Q4	8.00	8.35			
48Q1	6.00	7.70			
48Q2	7.00	8.00			
48Q3	6.00	7.16			
48Q4	8.00	6.09			
49Q1	4.00	6.95			
49Q2	4.00	6.00			
49Q3	3.00	5.05			
49Q4	5.00	6.03			
50Q1	4.00	4.30			
50Q2	7.00	5.82			
50Q3	8.00	7.31			
50Q4	11.00	9.58			
51Q1	8.00	10.03			
51Q2	12.00	10.24			
51Q3	12.00	11.84			
51Q4	14.00	12.94			
52Q1	10.00	11.20			
52Q2	9.00	12.32			
52Q3	9.00	9.67			
52Q4	10.00	10.77			
53Q1	9.00	8.40			
53Q2	10.00	9.41			
53Q3	10.00	10.65			
53Q4	11.00	11.24			
54Q1	9.00	10.93			
54Q2	11.00	10.75			
54Q3	10.00	11.60			
54Q4	15.00	11.54			
55Q1	11.00	13.29			
55Q2	13.00	12.86			
55Q3	16.00	12.67			
55Q4	24.00	17.76			
56Q1	17.00	18.34			
56Q2	22.00	17.02			
56Q3	23.00	20.81			
56Q4	26.00	24.07			
57Q1	18.00	18.33			
57Q2	20.00	19.46			
57Q3	18.00	18.74			
57Q4	19.00	18.54			
58Q1	17.00	13.61			
58Q2	15.00	17.58			
58Q3	11.00	14.31			
58Q4	12.00	13.37			
59Q1	13.00	12.54			
59Q2	17.00	13.28			
59Q3	21.00	14.92			
59Q4	18.00	20.17			
60Q1	18.00	18.31			
60Q2	20.00	19.91			

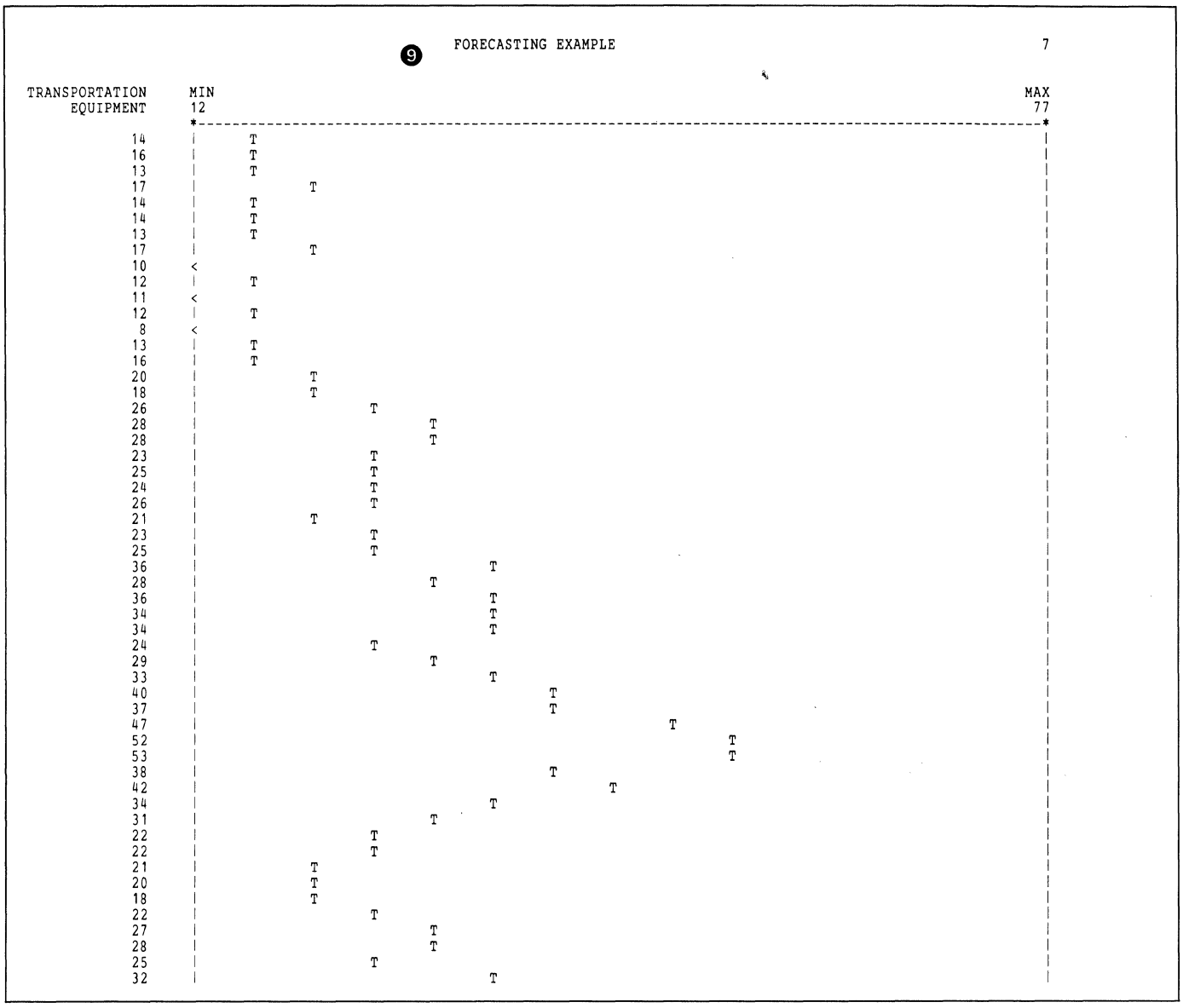
FORECASTING EXAMPLE 4

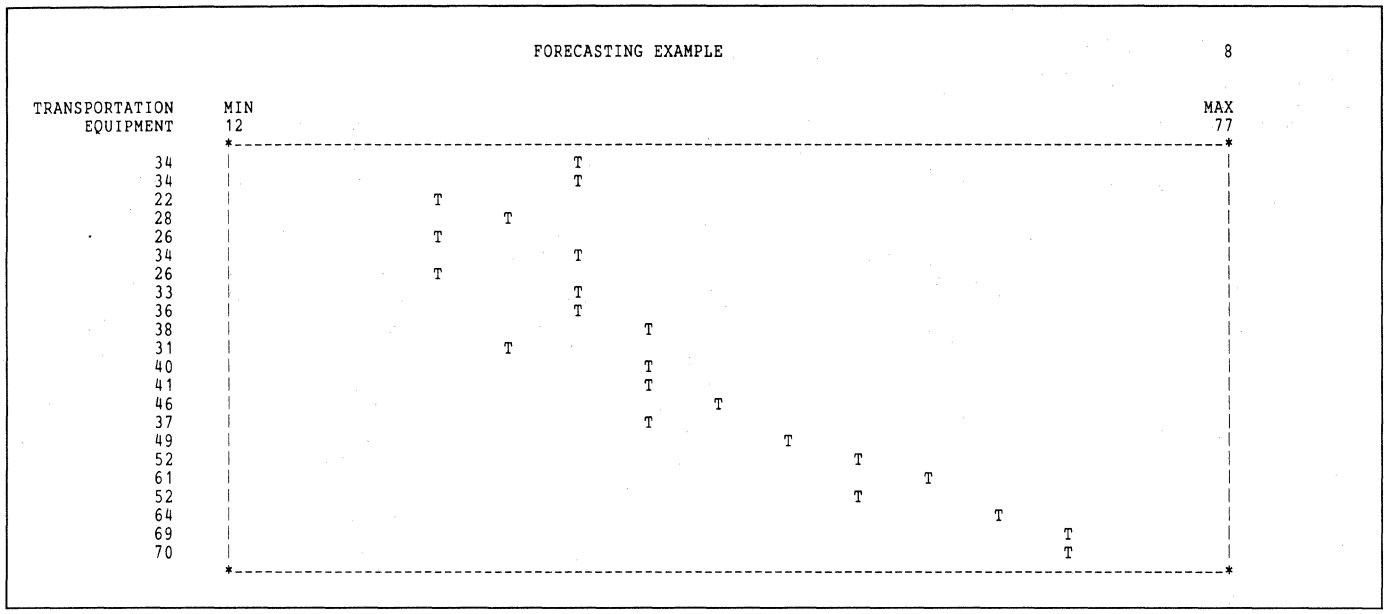
DATE	ACTUAL STONE CLAY GLASS	FORECAST STONE CLAY GLASS	L95 STONE CLAY GLASS	U95 STONE CLAY GLASS
60Q3	18.00	21.25		
60Q4	19.00	16.42		
61Q1	14.00	18.70		
61Q2	16.00	16.59		
61Q3	18.00	15.88		
61Q4	22.00	18.80		
62Q1	14.00	18.37		
62Q2	19.00	16.93		
62Q3	17.00	20.21		
62Q4	20.00	20.01		
63Q1	16.00	15.83		
63Q2	17.00	20.06		
63Q3	17.00	17.16		
63Q4	20.00	19.81		
64Q1	17.00	18.21		
64Q2	18.00	18.95		
64Q3	17.00	19.15		
64Q4	22.00	20.12		
65Q1	19.00	20.32		
65Q2	23.00	20.54		
65Q3	23.00	22.16		
65Q4	27.00	25.32		
66Q1		23.83	19.09	28.57
66Q2		25.48	19.89	31.08
66Q3		24.54	18.62	30.46
66Q4		26.07	20.02	32.12
67Q1		23.42	16.59	30.25
67Q2		24.29	17.36	31.22
67Q3		23.58	16.65	30.52
67Q4		24.48	17.53	31.42
68Q1		22.98	15.99	29.97
68Q2		23.92	16.90	30.94
68Q3		23.79	16.71	30.86
68Q4		24.60	17.48	31.72




FORECASTING EXAMPLE 5
8 8
TRANSPORTATION MIN MAX
EQUIPMENT 12 77
6

```
                              FORECASTING EXAMPLE                                6

 TRANSPORTATION          MIN                 MAX
      EQUIPMENT          12                   77
                         *--------------------*
              34         |     T              |
              34         |     T              |
              22         |   T                |
              28         |    T               |
              26         |   T                |
              34         |      T             |
              26         |   T                |
              33         |      T             |
              36         |      T             |
              38         |       T            |
              31         |    T               |
              40         |       T            |
              41         |       T            |
              46         |        T           |
              37         |       T            |
              49         |          T         |
              52         |           T        |
              61         |            T       |
              52         |           T        |
              64         |              T     |
              69         |               T    |
              70         |               T    |
                         *--------------------*
```


9
FORECASTING EXAMPLE 7
TRANSPORTATION EQUIPMENT
MIN 12
MAX 77
14
16
13
17
14
14
13
17
10
12
11
12
8
13
16
20
18
26
28
28
23
25
24
26
21
23
25
36
28
36
34
34
24
29
33
40
37
47
52
53
38
42
34
31
22
22
21
20
18
22
27
28
25
32



High-Low-Close Stock Market Reports: Example 3

The statements below produced the stock market example at the beginning of the chapter. This example overlays three plots on the same axes. The HILOC option causes the leftmost non-reference symbol (representing the low value of the Dow Jones Index) to be joined to the rightmost non-reference symbol (representing the high value). The reference line at MEAN (LOW) allows each day's values to be compared with the average low value of the index. The ID statement causes the values of DATE and VOLUME to be printed without being plotted.

```
DATA DOW;
   INPUT DATE DATE7. VOLUME HIGH LOW CLOSE;
   LABEL VOLUME='VOLUME IN THOUSANDS OF SHARES';
   FORMAT DATE DATE7.;
   CARDS;
03AUG81 3219.3 955.48 940.45 946.25
04AUG81 2938.5 951.39 937.40 945.97
05AUG81 4177.8 958.81 942.16 953.58
06AUG81 3975.7 961.47 947.30 952.91
07AUG81 3884.3 954.15 938.45 942.54
10AUG81 2937.7 948.82 935.88 943.68
11AUG81 5262.9 955.48 939.50 949.30
12AUG81 4005.2 955.86 942.26 945.21
13AUG81 3680.8 952.91 938.55 944.35
14AUG81 3714.1 947.77 933.79 936.93
17AUG81 3432.7 939.40 924.37 926.75
18AUG81 4396.7 932.74 916.38 924.37
19AUG81 3517.3 932.08 918.38 926.46
20AUG81 3811.9 935.31 923.52 928.37
21AUG81 2625.9 930.65 917.14 920.57
24AUG81 4736.1 917.43 896.97 900.11
25AUG81 4714.4 904.30 887.46 901.83
26AUG81 3279.6 908.39 893.65 899.26
27AUG81 3676.1 900.49 883.66 889.08
28AUG81 3024.2 898.78 884.80 892.22
;
```

```
PROC TIMEPLOT;
   PLOT LOW CLOSE HIGH/OVERLAY HILOC REF=MEAN(LOW);
   ID DATE VOLUME;
   FORMAT VOLUME 6.1 HIGH LOW CLOSE 6.2;
   TITLE 'HIGH-LOW-CLOSE STOCK MARKET DATA';
```

Chapter 53

The TRANSPOSE Procedure

Operating systems: All

ABSTRACT

The TRANSPOSE procedure transposes a SAS data set, changing observations into variables and vice versa.

INTRODUCTION

The TRANSPOSE procedure reads a SAS data set and creates a new data set as its only output. The rows of the original data matrix become columns, and columns become rows. Variables in the new data set correspond to observations in the old data set, and observations in the new data set correspond to variables in the old data set. TRANSPOSE also creates a variable in the new data set that contains the names of the variables from the old data set that were transposed.

For example,

```
DATA A;
   INPUT A B C;
   CARDS;
1 2 3
4 5 6
;
```

```
PROC TRANSPOSE;
PROC PRINT;
   TITLE 'SIMPLE TRANSPOSE EXAMPLE';
```

Output 53.1 A Transposed Data Set

SIMPLE TRANSPOSE EXAMPLE 1

OBS	_NAME_	COL1	COL2
1	A	1	4
2	B	2	5
3	C	3	6

PROC TRANSPOSE can be used with a BY statement to rearrange complex data sets in a variety of ways. In the following example, you have collected length and weight measurements for a sample of two species of fish, Mugil Curema (White Mullet) and Brevoortia Tyrannus (Atlantic Menhaden). During data collection it was convenient to enter the date and species once and string the measurements along the same line. Each line, therefore, represents a particular sample group and date.

What you need, however, are two data sets: one containing a separate observation for each length value, and one containing a separate observation for each weight value. Each data set must also indicate the group, species, and date corresponding to each length or weight measurement.

```
DATA FISHDATA;
   INPUT GROUP DATE DATE7. SPECIES $
         L1 W1 L2 W2 L3 W3 L4 W4;
   FORMAT DATE DATE7.;
   CARDS;
1  2JUN84 CUR 31 .25 32 .3  32 .25 33 .3
2  2JUN84 CUR 30 .20 36 .45 33 .25 36 .35
3  2JUN84 CUR 32 .35 32 .25 33 .30  .  .
1 11JUL84 MEN 72 4.0 82 5.5 70 4.0 79 4.3
1 14JUL84 MEN 79 4.1 78 3.8 80 3.8 74 3.6
1 13AUG84 MEN 86 70  83 54  89 76  80 49
;
PROC SORT;
   BY SPECIES DATE GROUP;
PROC PRINT;
   TITLE 'FISHDATA';

PROC TRANSPOSE DATA=FISHDATA OUT=LENGTH;
   BY SPECIES DATE GROUP;
   VAR L1-L4;
PROC PRINT;
   TITLE 'NEW DATA SET WITH LENGTH MEASUREMENTS';

PROC TRANSPOSE DATA=FISHDATA OUT=WEIGHT;
   BY SPECIES DATE GROUP;
   VAR W1-W4;
PROC PRINT;
   TITLE 'NEW DATA SET WITH WEIGHT MEASUREMENTS';
```

Output 53.2 PROC TRANSPOSE Converts the Data into More Usable Form

FISHDATA 1

OBS	GROUP	DATE	SPECIES	L1	W1	L2	W2	L3	W3	L4	W4
1	1	02JUN84	CUR	31	0.25	32	0.30	32	0.25	33	0.30
2	2	02JUN84	CUR	30	0.20	36	0.45	33	0.25	36	0.35
3	3	02JUN84	CUR	32	0.35	32	0.25	33	0.30	.	.
4	1	11JUL84	MEN	72	4.00	82	5.50	70	4.00	79	4.30
5	1	14JUL84	MEN	79	4.10	78	3.80	80	3.80	74	3.60
6	1	13AUG84	MEN	86	70.00	83	54.00	89	76.00	80	49.00

NEW DATA SET WITH LENGTH MEASUREMENTS 2

OBS	SPECIES	DATE	GROUP	_NAME_	COL1
1	CUR	02JUN84	1	L1	31
2	CUR	02JUN84	1	L2	32
3	CUR	02JUN84	1	L3	32
4	CUR	02JUN84	1	L4	33
5	CUR	02JUN84	2	L1	30
6	CUR	02JUN84	2	L2	36
7	CUR	02JUN84	2	L3	33
8	CUR	02JUN84	2	L4	36
9	CUR	02JUN84	3	L1	32
10	CUR	02JUN84	3	L2	32
11	CUR	02JUN84	3	L3	33
12	CUR	02JUN84	3	L4	.
13	MEN	11JUL84	1	L1	72
14	MEN	11JUL84	1	L2	82
15	MEN	11JUL84	1	L3	70
16	MEN	11JUL84	1	L4	79
17	MEN	14JUL84	1	L1	79
18	MEN	14JUL84	1	L2	78
19	MEN	14JUL84	1	L3	80
20	MEN	14JUL84	1	L4	74
21	MEN	13AUG84	1	L1	86
22	MEN	13AUG84	1	L2	83
23	MEN	13AUG84	1	L3	89
24	MEN	13AUG84	1	L4	80

NEW DATA SET WITH WEIGHT MEASUREMENTS 3

OBS	SPECIES	DATE	GROUP	_NAME_	COL1
1	CUR	02JUN84	1	W1	0.25
2	CUR	02JUN84	1	W2	0.30
3	CUR	02JUN84	1	W3	0.25
4	CUR	02JUN84	1	W4	0.30
5	CUR	02JUN84	2	W1	0.20
6	CUR	02JUN84	2	W2	0.45
7	CUR	02JUN84	2	W3	0.25
8	CUR	02JUN84	2	W4	0.35
9	CUR	02JUN84	3	W1	0.35
10	CUR	02JUN84	3	W2	0.25
11	CUR	02JUN84	3	W3	0.30
12	CUR	02JUN84	3	W4	.
13	MEN	11JUL84	1	W1	4.00
14	MEN	11JUL84	1	W2	5.50
15	MEN	11JUL84	1	W3	4.00
16	MEN	11JUL84	1	W4	4.30
17	MEN	14JUL84	1	W1	4.10
18	MEN	14JUL84	1	W2	3.80
19	MEN	14JUL84	1	W3	3.80
20	MEN	14JUL84	1	W4	3.60
21	MEN	13AUG84	1	W1	70.00
22	MEN	13AUG84	1	W2	54.00
23	MEN	13AUG84	1	W3	76.00
24	MEN	13AUG84	1	W4	49.00

SPECIFICATIONS

The statements used with PROC TRANSPOSE are

PROC TRANSPOSE *options*;
VAR *variables*;

ID *variable*;
IDLABEL *variable*;
COPY *variables*;
BY *variables*;

PROC TRANSPOSE Statement

PROC TRANSPOSE *options*;

These options can appear in the PROC TRANSPOSE statement:

DATA=*SASdataset*
: names the SAS data set to be transposed. If DATA= is omitted, the most recently created SAS data set is used.

PREFIX=*name*
: specifies a prefix to be used in constructing new variable names. For example, if PREFIX=VAR, the new variable names will be VAR1, VAR2,...,VAR*n*.

OUT=*SASdataset*
: specifies the name of the SAS data set that is created. If you want to create a permanent SAS data set, you must specify a two-level name. (See "SAS Files" for more information on permanent SAS data sets.) If OUT= is omitted, the SAS System still creates an output data set and automatically names it according to the DATA*n* convention, just as if you omitted a data set name in a DATA statement.

NAME=*name*
: provides a name for the variable in the output data set that contains the names of the transposed variables in the input data set. If NAME= is not specified, a variable is created automatically with the name _NAME_.

LABEL=*name*
: provides a name for the variable in the output data set that contains the labels of the transposed variables in the input data set. If LABEL= is not specified and at least one variable being transposed has a label, then a variable is created automatically with the name _LABEL_.

LET
: allows duplicate values of an ID variable within the data set or, if a BY statement is used, within a BY group. The observation containing the last occurrence of a particular ID value is transposed.

VAR Statement

VAR *variables*;

The VAR statement lists the variables to be transposed. Both numeric and character variables can be included.

If no VAR statement is used, all numeric variables in the old data set that are not listed in another statement are transposed. Character variables must be listed in a VAR statement if they are to be transposed. Variables that are not transposed are omitted from the new data set unless listed in the COPY or BY statements.

For example,

```
DATA B;
   INPUT A 1 B 3 X $5;
   CARDS;
```

```
1 2 X
3 4 Y
5 6 Z
;
PROC TRANSPOSE DATA=B;
PROC PRINT;
   TITLE 'NUMERIC VARIABLES TRANSPOSED BY DEFAULT';

PROC TRANSPOSE DATA=B;
   VAR X;
PROC PRINT;
   TITLE 'CHARACTER VARIABLE TRANSPOSED USING VAR STATEMENT';
```

Output 53.3 Using the VAR Statement

NUMERIC VARIABLES TRANSPOSED BY DEFAULT 1

OBS	_NAME_	COL1	COL2	COL3
1	A	.	.	.
2	B	.	.	.

CHARACTER VARIABLE TRANSPOSED USING VAR STATEMENT 2

OBS	_NAME_	COL1	COL2	COL3
1	X			

ID Statement

ID *variable*;

An ID statement causes the formatted values of the ID variable to be used as the names of the transposed variables in the new data set. Each formatted ID value should occur only once in the old data set, or, if a BY statement is used, only once within a BY group. Duplicate values cause PROC TRANSPOSE to issue an error message and stop unless the LET option is specified.

TRANSPOSE will change a formatted ID value into a valid SAS name whenever necessary. If the formatted value looks like a numeric constant, TRANSPOSE changes the characters "+", "−", and "." to "P","N", and "D", respectively. If the first character is numeric, an underscore is prefixed to the value, truncating the last character of an 8-character value. Any remaining invalid characters are replaced by underscores.

If an ID statement is used, observations with missing ID values are omitted from the new data set. For example,

```
DATA C;
   INPUT A 1 B 3 X $5;
   CARDS;
1 2 X
3 4 Y
5 6 Z
7 8
;
```

```
PROC TRANSPOSE;
   ID X;
PROC PRINT;
   TITLE 'TRANSPOSING WITH AN ID STATEMENT';
```

Output 53.4 Using the ID Statement

```
                    TRANSPOSING WITH AN ID STATEMENT                    1

                          OBS    _NAME_

                           1       A
                           2       B
```

IDLABEL Statement

IDLABEL *variable;*
IDL *variable;*

If an ID statement is present, the IDLABEL statement can also be used to specify a character or numeric variable to provide labels for the transposed variables in the new data set.

COPY Statement

COPY *variables;*

All variables in the COPY statement are copied directly from the input data set to the output data set without being transposed. The output data set is padded with missing values if necessary. For example,

```
DATA D;
   INPUT A B C;
   CARDS;
1  2  3
4  5  6
7  8  9
11 22 33
;
PROC TRANSPOSE;
   COPY C;
PROC PRINT;
   TITLE 'TRANSPOSING WITH A COPY STATEMENT';
```

Output 53.5 Using the COPY Statement

```
                    TRANSPOSING WITH A COPY STATEMENT                   1

 OBS     C     _NAME_    COL1    COL2    COL3    COL4

  1      3       A        1       4       7       11
  2      6       B        2       5       8       22
  3      9                .       .       .        .
  4     33                .       .       .        .
```

BY Statement

BY *variables;*

When a BY statement is used, PROC TRANSPOSE transposes each BY group. The BY variables are included in the output data set but are not transposed.

```
DATA E;
   INPUT B X $ Y $;
   CARDS;
1 X1 Y1
2 X2 Y2
3 X3 Y3
;
PROC TRANSPOSE;
   VAR X Y;
   BY B;
PROC PRINT;
   TITLE 'TRANSPOSING WITH A BY STATEMENT';
```

Output 53.6 Using the BY Statement

TRANSPOSING WITH A BY STATEMENT 1

OBS	B	_NAME_	COL1
1	1	X	X1
2	1	Y	Y1
3	2	X	X2
4	2	Y	Y2
5	3	X	X3
6	3	Y	Y3

When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING.

For more information, see the discussion of the BY statement in "Statements Used in the PROC Step" and in **Using a BY Statement,** below.

DETAILS

Output Data Set

Contents The new data set has a character variable that contains the variable names from the old data set. The name of the variable is specified by the NAME= option. If NAME= is not specified, a variable is created automatically and named _NAME_. If the old data set has variable labels for the transposed variables, the labels are placed in a character variable in the new data set. The name of the variable is specified by the LABEL= option. If LABEL= is not specified, a variable is created automatically and named _LABEL_. However, if the variables referenced by the NAME= or LABEL= options (or, if these options are omitted, variables called _NAME_ or _LABEL_), are listed in the COPY or BY statements, then PROC TRANSPOSE does not create variables with duplicate names containing the names or labels of the transposed input variables.

All transposed output variables are the same type and length. If all input variables to be transposed are numeric, then the transposed output variables are numeric. If **any** such input variable is character, then **all** transposed output variables are character. The character values of transposed numeric variables in the output data set are the formatted values of the input numeric variables. The length of the transposed output variables is equal to the length of the longest transposed input variable.

All variables listed in the COPY statement are copied to the new data set.

Variable names There are four ways that the transposed variables in the new data set can be named:

1. by an ID statement specifying the name of a variable in the old data set. The formatted values of this variable are used as names for the output variables.
2. by the PREFIX= option on the PROC TRANSPOSE statement. For example, if you request PREFIX=VAR, the new variable names will be VAR1, VAR2,...,VAR*n*.
3. by an input variable _NAME_. If you use neither the ID statement nor the PREFIX= option, PROC TRANSPOSE looks for an input variable called _NAME_ from which to get the output variable names. The variable _NAME_ can be the result of a previous execution of PROC TRANSPOSE. If so, the variable names are identical to the variable names in the original (untransposed) data set.
4. by default. If you do not use an ID statement or the PREFIX= option, and there is no input variable called _NAME_, PROC TRANSPOSE assigns the names COL1, COL2,..., COL*n* to the variables in the output data set.

Using a BY Statement

If a BY statement is used with PROC TRANSPOSE, there is one observation corresponding to each transposed variable of the old data set in each BY group in the new data set. The method of naming the output variables determines what output variables appear in the new data set.

If the output variables are named by the PREFIX= option or by the default prefix COL, the number of transposed output variables is the maximum number of observations in any BY group of the old data set. If a BY group in the old data set has fewer than the maximum number of observations, then the BY group in the new data set has missing values for the variables with no corresponding input observations. The following example shows the use of a BY statement and the PREFIX= option with the data set FISHDATA:

```
PROC SORT DATA=FISHDATA;
   BY GROUP;
PROC PRINT DATA=FISHDATA;
   TITLE 'FISHDATA: ORIGINAL DATA SET';

PROC TRANSPOSE DATA=FISHDATA OUT=TRAN PREFIX=VAR;
   VAR L1-L4 W1-W4;
   BY GROUP;
PROC PRINT DATA=TRAN;
   TITLE 'FISHDATA TRANSPOSED WITH BY GROUPS USING THE
      PREFIX=VAR OPTION';
```

Output 53.7 FISHDATA is Transposed and the Output Data Set is Called TRAN

FISHDATA: ORIGINAL DATA SET 1

OBS	GROUP	DATE	SPECIES	L1	W1	L2	W2	L3	W3	L4	W4
1	1	02JUN84	CUR	31	0.25	32	0.30	32	0.25	33	0.30
2	1	11JUL84	MEN	72	4.00	82	5.50	70	4.00	79	4.30
3	1	14JUL84	MEN	79	4.10	78	3.80	80	3.80	74	3.60
4	1	13AUG84	MEN	86	70.00	83	54.00	89	76.00	80	49.00
5	2	02JUN84	CUR	30	0.20	36	0.45	33	0.25	36	0.35
6	3	02JUN84	CUR	32	0.35	32	0.25	33	0.30	.	.

FISHDATA TRANSPOSED WITH BY GROUPS USING THE PREFIX=VAR OPTION 2

OBS	GROUP	_NAME_	VAR1	VAR2	VAR3	VAR4
1	1	L1	31.00	72.0	79.0	86
2	1	L2	32.00	82.0	78.0	83
3	1	L3	32.00	70.0	80.0	89
4	1	L4	33.00	79.0	74.0	80
5	1	W1	0.25	4.0	4.1	70
6	1	W2	0.30	5.5	3.8	54
7	1	W3	0.25	4.0	3.8	76
8	1	W4	0.30	4.3	3.6	49
9	2	L1	30.00	.	.	.
10	2	L2	36.00	.	.	.
11	2	L3	33.00	.	.	.
12	2	L4	36.00	.	.	.
13	2	W1	0.20	.	.	.
14	2	W2	0.45	.	.	.
15	2	W3	0.25	.	.	.
16	2	W4	0.35	.	.	.
17	3	L1	32.00	.	.	.
18	3	L2	32.00	.	.	.
19	3	L3	33.00	.	.	.
20	3	L4	.	.	.	.
21	3	W1	0.35	.	.	.
22	3	W2	0.25	.	.	.
23	3	W3	0.30	.	.	.
24	3	W4	.	.	.	.

If an ID statement is used so that the output names are taken from an input variable, there is one output variable for each distinct value of the ID variable. The output variables are ordered according to the first appearance of their names in the old data set. If a value for the ID variable does not appear in a BY group, the corresponding output variable has missing values in that BY group. For example, when the FISHDATA data set is transposed with an ID statement in the program:

```
PROC TRANSPOSE DATA=FISHDATA OUT=TRAN2;
   ID DATE;
   BY GROUP;
PROC PRINT;
   TITLE 'FISHDATA TRANSPOSED WITH BY GROUPS USING AN ID STATEMENT';
```

the output data set looks like this:

Output 53.8 FISHDATA is Transposed and the Output Data Set is Called TRAN2

FISHDATA TRANSPOSED WITH BY GROUPS USING AN ID STATEMENT 1

OBS	GROUP	_NAME_	_02JUN84	_11JUL84	_14JUL84	_13AUG84
1	1	L1	31.00	72.0	79.0	86
2	1	W1	0.25	4.0	4.1	70
3	1	L2	32.00	82.0	78.0	83
4	1	W2	0.30	5.5	3.8	54
5	1	L3	32.00	70.0	80.0	89
6	1	W3	0.25	4.0	3.8	76
7	1	L4	33.00	79.0	74.0	80
8	1	W4	0.30	4.3	3.6	49
9	2	L1	30.00	.	.	.
10	2	W1	0.20	.	.	.
11	2	L2	36.00	.	.	.
12	2	W2	0.45	.	.	.
13	2	L3	33.00	.	.	.
14	2	W3	0.25	.	.	.
15	2	L4	36.00	.	.	.
16	2	W4	0.35	.	.	.
17	3	L1	32.00	.	.	.
18	3	W1	0.35	.	.	.
19	3	L2	32.00	.	.	.
20	3	W2	0.25	.	.	.
21	3	L3	33.00	.	.	.
22	3	W3	0.30	.	.	.
23	3	L4	.	.	.	.
24	3	W4	.	.	.	.

EXAMPLES

Transposing a Data Set Twice: Example 1

This example shows that transposing a data set twice can produce a data set similar to the original data set, except for the new variable _NAME_.

```
DATA A;
   INPUT A B C;
   CARDS;
1 2 3
4 5 6
7 8 9
10 11 12
;
PROC PRINT;
   TITLE 'BEFORE TRANSPOSING';

PROC TRANSPOSE;
PROC PRINT;
   TITLE 'AFTER TRANSPOSING ONCE';

PROC TRANSPOSE;
PROC PRINT;
   TITLE 'AFTER TRANSPOSING TWICE';
```

Output 53.9 Transposing a Data Set Twice

BEFORE TRANSPOSING 1

OBS	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9
4	10	11	12

AFTER TRANSPOSING ONCE 2

OBS	_NAME_	COL1	COL2	COL3	COL4
1	A	1	4	7	10
2	B	2	5	8	11
3	C	3	6	9	12

AFTER TRANSPOSING TWICE 3

OBS	_NAME_	A	B	C
1	COL1	1	2	3
2	COL2	4	5	6
3	COL3	7	8	9
4	COL4	10	11	12

Converting a List to a Table: Example 2

The data below are a list of cities with temperatures in January, July, or August. The desired output is a table with cities as rows and two columns showing minimum winter and maximum summer temperatures. The cities should be ordered from coldest to hottest.

```
DATA LIST;
   INPUT CITY $15. MONTH $15. TEMP;
   CARDS;
RALEIGH         JULY            77.5
RALEIGH         JANUARY         40.5
MIAMI           AUGUST          82.9
MIAMI           JANUARY         67.2
LOS_ANGELES     AUGUST          69.5
LOS_ANGELES     JANUARY         54.5
JUNEAU          JULY            55.7
JUNEAU          JANUARY         23.5
PHOENIX         JULY            91.2
PHOENIX         JANUARY         51.2
BISMARCK        JANUARY          8.2
BISMARCK        JULY            70.8
CHICAGO         AUGUST          71.1
CHICAGO         JANUARY         22.9
WICHITA         JULY            80.7
WICHITA         JANUARY         31.3
HONOLULU        AUGUST          80.7
HONOLULU        JANUARY         72.3
BOSTON          JULY            73.3
BOSTON          JANUARY         29.2
DULUTH          JULY            65.6
DULUTH          JANUARY          8.5
;
```

```
PROC SORT;
   BY CITY;
PROC PRINT;
   TITLE 'ORIGINAL DATA IN LIST FORM';

PROC TRANSPOSE;
   BY CITY;
   ID MONTH;
DATA; SET;
   WINTER=JANUARY;
   SUMMER=MAX(JULY,AUGUST);
   MEAN=MEAN(JANUARY,JULY,AUGUST);
PROC SORT;
   BY MEAN;
PROC PRINT;
   TITLE 'TABULAR DATA';
   ID CITY;
   VAR WINTER SUMMER;
```

Output 53.10 Converting Data in List Form to Tabular Form

ORIGINAL DATA IN LIST FORM 1

OBS	CITY	MONTH	TEMP
1	BISMARCK	JANUARY	8.2
2	BISMARCK	JULY	70.8
3	BOSTON	JULY	73.3
4	BOSTON	JANUARY	29.2
5	CHICAGO	AUGUST	71.1
6	CHICAGO	JANUARY	22.9
7	DULUTH	JULY	65.6
8	DULUTH	JANUARY	8.5
9	HONOLULU	AUGUST	80.7
10	HONOLULU	JANUARY	72.3
11	JUNEAU	JULY	55.7
12	JUNEAU	JANUARY	23.5
13	LOS_ANGELES	AUGUST	69.5
14	LOS_ANGELES	JANUARY	54.5
15	MIAMI	AUGUST	82.9
16	MIAMI	JANUARY	67.2
17	PHOENIX	JULY	91.2
18	PHOENIX	JANUARY	51.2
19	RALEIGH	JULY	77.5
20	RALEIGH	JANUARY	40.5
21	WICHITA	JULY	80.7
22	WICHITA	JANUARY	31.3

TABULAR DATA 2

CITY	WINTER	SUMMER
DULUTH	8.5	65.6
BISMARCK	8.2	70.8
JUNEAU	23.5	55.7
CHICAGO	22.9	71.1
BOSTON	29.2	73.3
WICHITA	31.3	80.7
RALEIGH	40.5	77.5
LOS_ANGELES	54.5	69.5
PHOENIX	51.2	91.2
MIAMI	67.2	82.9
HONOLULU	72.3	80.7

Rearranging Data from a Factorial Design: Example 3

The data below are from a subject × drug × exercise factorial design. This example shows how PROC TRANSPOSE can rearrange a more complicated data set using BY groups and ID variables.

```
DATA A1;
   INPUT SUBJECT $ DRUG $ EXERCISE $ RESPONSE;
   CARDS;
SMITH     ASPIRIN    LIGHT      5
SMITH     ASPIRIN    MEDIUM     8
SMITH     ASPIRIN    HEAVY      9
SMITH     PLACEBO    LIGHT      4
SMITH     PLACEBO    HEAVY      7
JONES     ASPIRIN    LIGHT      6
JONES     ASPIRIN    MEDIUM     7
JONES     ASPIRIN    HEAVY      9
JONES     PLACEBO    LIGHT      3
JONES     PLACEBO    MEDIUM     4
JONES     PLACEBO    HEAVY      6
;
PROC SORT;
   BY SUBJECT DRUG;
PROC PRINT;
   BY SUBJECT;
   TITLE 'DATA SET A1';

PROC TRANSPOSE OUT=A2 NAME=RESPNAME;
   BY SUBJECT DRUG;
   ID EXERCISE;
   VAR RESPONSE;
PROC PRINT;
   BY SUBJECT;
   TITLE 'DATA SET A2';

PROC TRANSPOSE OUT=A3 NAME=EXERCISE;
   BY SUBJECT RESPNAME;
   ID DRUG;
   VAR LIGHT MEDIUM HEAVY;
PROC PRINT;
   BY SUBJECT;
   TITLE 'DATA SET A3';

PROC TRANSPOSE OUT=A4 NAME=DRUG;
   BY SUBJECT EXERCISE NOTSORTED;
   ID RESPNAME;
PROC PRINT;
   BY SUBJECT;
   TITLE 'DATA SET A4';
```

Output 53.11 Data from a Subject-by-Drug-by-Exercise Factorial Design

DATA SET A1 1

SUBJECT=JONES

OBS	DRUG	EXERCISE	RESPONSE
1	ASPIRIN	LIGHT	6
2	ASPIRIN	MEDIUM	7
3	ASPIRIN	HEAVY	9
4	PLACEBO	LIGHT	3
5	PLACEBO	MEDIUM	4
6	PLACEBO	HEAVY	6

SUBJECT=SMITH

OBS	DRUG	EXERCISE	RESPONSE
7	ASPIRIN	LIGHT	5
8	ASPIRIN	MEDIUM	8
9	ASPIRIN	HEAVY	9
10	PLACEBO	LIGHT	4
11	PLACEBO	HEAVY	7

DATA SET A2 2

SUBJECT=JONES

OBS	DRUG	RESPNAME	LIGHT	MEDIUM	HEAVY
1	ASPIRIN	RESPONSE	6	7	9
2	PLACEBO	RESPONSE	3	4	6

SUBJECT=SMITH

OBS	DRUG	RESPNAME	LIGHT	MEDIUM	HEAVY
3	ASPIRIN	RESPONSE	5	8	9
4	PLACEBO	RESPONSE	4	.	7

DATA SET A3 3

SUBJECT=JONES

OBS	RESPNAME	EXERCISE	ASPIRIN	PLACEBO
1	RESPONSE	LIGHT	6	3
2	RESPONSE	MEDIUM	7	4
3	RESPONSE	HEAVY	9	6

SUBJECT=SMITH

OBS	RESPNAME	EXERCISE	ASPIRIN	PLACEBO
4	RESPONSE	LIGHT	5	4
5	RESPONSE	MEDIUM	8	.
6	RESPONSE	HEAVY	9	7

DATA SET A4 4

SUBJECT=JONES

OBS	EXERCISE	DRUG	RESPONSE
1	LIGHT	ASPIRIN	6
2	LIGHT	PLACEBO	3
3	MEDIUM	ASPIRIN	7
4	MEDIUM	PLACEBO	4
5	HEAVY	ASPIRIN	9
6	HEAVY	PLACEBO	6

(continued on next page)

(continued from previous page)

```
-------------------------------------------------SUBJECT=SMITH-------------------------------------------------

                          OBS    EXERCISE     DRUG      RESPONSE

                            7     LIGHT      ASPIRIN       5
                            8     LIGHT      PLACEBO       4
                            9     MEDIUM     ASPIRIN       8
                           10     MEDIUM     PLACEBO       .
                           11     HEAVY      ASPIRIN       9
                           12     HEAVY      PLACEBO       7
```

Transposing a Large Data Set: Example 4

The following example shows how to transpose a data set that is too large to be held in core. A small data set is used for illustrative purposes.

```
DATA FIRST;
   ARRAY X X1-X5;
   DO N=1 TO 10;
      DO OVER X;
         X=N*100+_I_;
         END;
      OUTPUT;
      END;
PROC PRINT;
   VAR N X1-X5;
   TITLE 'ORIGINAL DATA SET';

PROC TRANSPOSE OUT=TEMP;
   BY N;
PROC PRINT;
   VAR N _NAME_ COL1;
   TITLE 'INTERMEDIATE DATA SET BEFORE SORTING';

PROC SORT OUT=TEMP;
   BY _NAME_;
PROC PRINT;
   VAR N _NAME_ COL1;
   TITLE 'INTERMEDIATE DATA SET AFTER SORTING';

PROC TRANSPOSE DATA=TEMP OUT=FINAL;
   VAR COL1;
   BY _NAME_;
PROC PRINT;
   TITLE 'FINAL TRANSPOSED DATA SET';
```

Output 53.12 Transposing a Large Data Set

ORIGINAL DATA SET 1

OBS	N	X1	X2	X3	X4	X5
1	1	101	102	103	104	105
2	2	201	202	203	204	205
3	3	301	302	303	304	305
4	4	401	402	403	404	405
5	5	501	502	503	504	505
6	6	601	602	603	604	605
7	7	701	702	703	704	705
8	8	801	802	803	804	805
9	9	901	902	903	904	905
10	10	1001	1002	1003	1004	1005

INTERMEDIATE DATA SET BEFORE SORTING 2

OBS	N	_NAME_	COL1
1	1	X1	101
2	1	X2	102
3	1	X3	103
4	1	X4	104
5	1	X5	105
6	2	X1	201
7	2	X2	202
8	2	X3	203
9	2	X4	204
10	2	X5	205
11	3	X1	301
12	3	X2	302
13	3	X3	303
14	3	X4	304
15	3	X5	305
16	4	X1	401
17	4	X2	402
18	4	X3	403
19	4	X4	404
20	4	X5	405
21	5	X1	501
22	5	X2	502
23	5	X3	503
24	5	X4	504
25	5	X5	505
26	6	X1	601
27	6	X2	602
28	6	X3	603
29	6	X4	604
30	6	X5	605
31	7	X1	701
32	7	X2	702
33	7	X3	703
34	7	X4	704
35	7	X5	705
36	8	X1	801
37	8	X2	802
38	8	X3	803
39	8	X4	804
40	8	X5	805
41	9	X1	901
42	9	X2	902
43	9	X3	903
44	9	X4	904
45	9	X5	905
46	10	X1	1001
47	10	X2	1002
48	10	X3	1003
49	10	X4	1004
50	10	X5	1005

INTERMEDIATE DATA SET AFTER SORTING 3

OBS	N	_NAME_	COL1
1	1	X1	101
2	2	X1	201
3	3	X1	301
4	4	X1	401
5	5	X1	501
6	6	X1	601
7	7	X1	701
8	8	X1	801
9	9	X1	901
10	10	X1	1001
11	1	X2	102
12	2	X2	202
13	3	X2	302
14	4	X2	402
15	5	X2	502
16	6	X2	602
17	7	X2	702
18	8	X2	802
19	9	X2	902
20	10	X2	1002
21	1	X3	103
22	2	X3	203
23	3	X3	303
24	4	X3	403
25	5	X3	503
26	6	X3	603
27	7	X3	703
28	8	X3	803
29	9	X3	903
30	10	X3	1003
31	1	X4	104
32	2	X4	204
33	3	X4	304
34	4	X4	404
35	5	X4	504
36	6	X4	604
37	7	X4	704
38	8	X4	804
39	9	X4	904
40	10	X4	1004
41	1	X5	105
42	2	X5	205
43	3	X5	305
44	4	X5	405
45	5	X5	505
46	6	X5	605
47	7	X5	705
48	8	X5	805
49	9	X5	905
50	10	X5	1005

FINAL TRANSPOSED DATA SET 4

OBS	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	X1	101	201	301	401	501	601	701	801	901	1001
2	X2	102	202	302	402	502	602	702	802	902	1002
3	X3	103	203	303	403	503	603	703	803	903	1003
4	X4	104	204	304	404	504	604	704	804	904	1004
5	X5	105	205	305	405	505	605	705	805	905	1005

Chapter 54

The UNIVARIATE Procedure

Operating systems: All

ABSTRACT

The UNIVARIATE procedure produces simple descriptive statistics (including quantiles) for numeric variables.

INTRODUCTION

The UNIVARIATE procedure differs from other SAS procedures that produce descriptive statistics because it provides greater detail on the distribution of a variable. Features in PROC UNIVARIATE include:

- detail on the extreme values of a variable
- quantiles, such as the median
- several plots to picture the distribution
- frequency tables

- a test that the data are normally distributed.

If a BY statement is used with PROC UNIVARIATE, descriptive statistics are calculated separately for groups of observations. PROC UNIVARIATE can also create one or more data sets containing calculated statistics.

SPECIFICATIONS

The following statements control the UNIVARIATE procedure:

PROC UNIVARIATE *options*;
VAR *variables*;
BY *variables*;
FREQ *variable*;
WEIGHT *variable*;
ID *variables*;
OUTPUT OUT=*SASdataset keyword=names...*;

Several OUTPUT statements are permitted, but only one each of the other statements. The statements after the PROC statement can be listed in any order.

PROC UNIVARIATE Statement

PROC UNIVARIATE *options*;

The options that can appear in the PROC UNIVARIATE statement are listed below:

DATA=*SASdataset*
: names the SAS data set to be used by PROC UNIVARIATE. If DATA= is omitted, the most recently created SAS data set is used.

NOPRINT
: suppresses all printed output. NOPRINT can be used when the only purpose for executing the procedure is to create new data sets.

PLOT
: causes PROC UNIVARIATE to produce a stem-and-leaf plot (or a horizontal bar chart), a box plot, and a normal probability plot.

FREQ
: requests a frequency table consisting of the variable values, frequencies, percentages, and cumulative percentages.

NORMAL
: causes PROC UNIVARIATE to compute a test statistic for the hypothesis that the input data come from a normal distribution. The probability of a more extreme value of the test statistic is also printed.

PCTLDEF=*value*
: specifies which of the four definitions presented in **Computational Methods** is to be used to calculate percentiles. The PCTLDEF= value can be 1, 2, 3, 4, or 5. If PCTLDEF= is omitted, definition 4 is used.

VARDEF=DF
VARDEF=WEIGHT | WGT
VARDEF=N
VARDEF=WDF
: specifies the divisor to be used in the calculation of the variances. VARDEF=DF requests that the degrees of freedom (N−1) be used

as the divisor. VARDEF=WEIGHT requests that the sum of the weights be used. VARDEF=N requests that the number of observations (N) be used. VARDEF=WDF requests that the sum of the weights minus one be used. The default is VARDEF=DF.

VAR Statement

VAR *variables*;

Univariate descriptive measures are calculated for all numeric variables listed in the VAR statement. If no VAR statement appears, all numeric variables in the data set are analyzed. A VAR statement must be included when an OUTPUT statement is used.

BY Statement

BY *variables*;

A BY statement can be used with PROC UNIVARIATE to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If your input data set is not sorted in ascending order, use the SORT procedure with a similar BY statement to sort the data, or, if appropriate, use the BY statement options NOTSORTED or DESCENDING. For more information, see the discussion of the BY statement in "Statements Used in the PROC Step."

FREQ Statement

FREQ *variable*;

When a FREQ statement appears, each observation in the data set being analyzed is assumed to represent n observations, where n is the value of the FREQ variable. If the FREQ variable has a value that is less than one, the observation is not used in the analysis. If the value is not an integer, only the integer portion is used.

The statistics calculated using a FREQ statement are identical to an analysis produced using a data set that contains n observations in place of each observation in the input data set.

WEIGHT Statement

WEIGHT *variable*;

When a WEIGHT statement is specified, PROC UNIVARIATE uses the value of the WEIGHT variable, w_i, to calculate a weighted mean $\bar{x}_w$ and a weighted variance s_w^2 as

$$\bar{x}_w = \Sigma_i w_i x_i / \Sigma_i w_i$$

and

$$s_w^2 = \Sigma_i w_i (x_i - \bar{x}_w)^2 / d$$

where the x_i are the variable values and the divisor d is controlled by the VARDEF= option. The divisor can be $n-1$ (VARDEF=DF), Σw_i (VARDEF=WEIGHT | WGT), n (VARDEF=N) or Σw_i-1 (VARDEF=WDF) where n is the number of values.

If the value of the WEIGHT variable is less than zero, then a value of zero for the weight is assumed.

The value of the WEIGHT variable is used only to calculate the first two sample moments and related statistics. Hence, if a WEIGHT statement is used, measures of skewness and kurtosis are not calculated and are reported as missing values. The WEIGHT variable has no effect on the calculation of quantiles or extremes.

ID Statement

ID *variables;*

When an ID statement is used, up to eight characters of the first variable specified are used to identify observations in the printed listing of the five largest and five smallest values. However, the *values* of these ID variables are written to any OUTPUT data set specified in their entirety. In addition, if one or more OUTPUT statements are used, the values of the ID variables are placed in each output data set. The values of the ID variables used in the output data set are taken either from the first observation in the data set analyzed by PROC UNIVARIATE or from the first observation in the current BY group, if a BY statement is used.

OUTPUT Statement

OUTPUT OUT=*SASdataset keyword=names...;*

The OUTPUT statement requests that PROC UNIVARIATE output statistics to a new SAS data set. The options name the new data set and specify the variables to be included.

OUT=*SASdataset*
: names the output data set. If you want to create a permanent SAS data set, you must specify a two-level name (see "SAS Files" for more information on permanent data sets).

keyword=names
: specifies the statistics you want in the new data set and names the new variables that contain the statistics. Write the keyword for the desired statistic, an equal sign, and the variable or variables to contain the statistic.

  In the output data set, the first variable listed after a keyword in the OUTPUT statement contains that statistic for the first variable listed in the VAR statement; the second variable contains the statistic for the second variable in the VAR statement, and so on.

  The list of variables following the equal sign can be shorter than the list of variables in the VAR statement.

The valid keywords and the statistics they represent are as follows:

N	the number of observations on which the calculations are based.
NMISS	the number of missing values.
NOBS	the number of observations.
MEAN	the mean.
SUM	the sum.
STD	the standard deviation.
VAR	the variance.
SKEWNESS	skewness.
KURTOSIS	kurtosis.

SUMWGT	the sum of the weights.
MAX	the largest value.
MIN	the smallest value.
RANGE	the range.
Q3	the upper quartile or the seventy-fifth percentile.
MEDIAN	the median or the fiftieth percentile.
Q1	the lower quartile or the twenty-fifth percentile.
QRANGE	the difference between the upper and lower quartiles, that is, Q3-Q1.
P1	the first percentile.
P5	the fifth percentile.
P10	the tenth percentile.
P90	the ninetieth percentile.
P95	the ninety-fifth percentile.
P99	the ninety-ninth percentile.
MODE	the most frequent value. If there is more than one mode, the smallest mode is used.
SIGNRANK	the signed rank statistic.
NORMAL	the test statistic for normality. If the sample size is less than fifty-one this is the Shapiro-Wilk statistic. Otherwise, it is the Kolmogorov statistic.

The number of observations in the new data set corresponds to the number of groups for which statistics are calculated. The variables in the BY statement and the ID statement, as well as the computed statistics, are included in the new data set.

For example, consider these statements:

```
PROC UNIVARIATE;
   VAR GRADE1 GRADE2;
   BY SEX;
   OUTPUT OUT=NEW MEAN=AVE1 AVE2 VAR=VAR1;
```

If the BY variable SEX has two values, F and M, the data set NEW contains two observations. Each of these observations contains the variables SEX, AVE1, AVE2, and VAR1.

Any number of OUTPUT statements can be used with each execution of PROC UNIVARIATE.

DETAILS

Missing Values

If a variable for which statistics are to be calculated has a missing value, that value is ignored in the calculation of statistics and the missing values are tabulated separately. A missing value for one such variable does not affect the treatment of other variables in the same observation.

If the WEIGHT variable has a missing value, the weight is taken to be zero. However, the observation is still used to calculate quantiles and extremes. If the FREQ variable has a missing value, the observation is not used at all.

If a variable in a BY or ID statement has a missing value, the procedure treats it as it would treat any other value of a BY or ID variable.

Output Data Set

If an OUTPUT statement is used, the corresponding output data set contains an observation for each unique set of values of the variables in the BY statement, or a single observation if there is no BY statement. The variables in each observation consist of the variables in the BY statement, the variables in the ID statement, and a selection of the statistics from the list above as selected in the OUTPUT statement.

The values of the variables listed in the BY statement are taken as those of the corresponding BY group, the values of the ID variables are taken from the first observation of each BY group, and the values of the statistics are computed from the values of the variables within each BY group or across all the data if there is no BY statement.

Computational Methods

The sample mean, the sample standard deviation, the minimum, and the maximum are computed using the original data. All other statistics are computed after the data have been truncated to single precision (approximately seven significant digits).

Standard algorithms (Fisher 1973) are used to compute the moment statistics. Using the PCTLDEF= option, you can specify one of five methods for computing quantile statistics.

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. For the tth percentile, where $p=t/100$, let

$$np = j + g$$

where j is the integer part and g is the fractional part of np.

The tth percentile, y, for example, is defined as:

DEFINITION 1: weighted average at x_{np}

$$y = (1-g)x_j + gx_{j+1}$$

where x_0 is taken to be x_1

DEFINITION 2: observation numbered closest to np

$$y = x_i$$

where i is the integer part of $np + 1/2$

DEFINITION 3: empirical distribution function

$$y = x_j \quad \text{if } g = 0$$
$$y = x_{j+1} \text{ if } g > 0$$

DEFINITION 4: weighted average aimed at $x_{p(n+1)}$

$y = (1-g)x_j + gx_{j+1}$

where $(n+1)p = j + g$

where x_{n+1} is taken to be x_n

DEFINITION 5: empirical distribution function with averaging

$y=(x_j+x_{j+1})/2$ if $g=0$

$y=x_{j+1}$ if $g>0$

where $np=j+g$.

The signed rank statistic S is computed as

$$S = \Sigma r_i^+ - n(n+1)/4$$

where r_i^+ is the rank of $|x_i|$ and n is the number of nonzero x_i values. Average ranks are used for tied values. The significance of the S is computed by treating $(|S|-.5)/V^{**}.5$ as a standard normal deviate where V is computed as

$$V = (n(n+1)(2n+1) - .5\Sigma t_i(t_i+1)(t_i-1))/24$$

where the sum is over groups tied in absolute value and t_i is the number of values in the ith group (Lehmann 1975).

Test of Normality

When the NORMAL option is specified in the PROC UNIVARIATE statement, the procedure produces a test statistic for the null hypothesis that the input data values are a random sample from a normal distribution.

If the sample size is less than fifty-one, the Shapiro-Wilk statistic, W, is computed. The W statistic is the ratio of the best estimator of the variance (based on the square of a linear combination of the order statistics) to the usual corrected sum of squares estimator of the variance. W must be greater than zero and less than or equal to one, with small values of W leading to rejection of the null hypothesis. The computed value of W is used to interpolate linearly within the range of simulated critical values given in Shapiro and Wilk (1965).

If the sample size is greater than fifty, the data are tested against a normal distribution with mean and variance equal to the sample mean and variance. The usual Kolomogorov D statistic is computed and printed. The probability of a larger test statistic is obtained by forming the value

$$(\sqrt{n} - .01 + .85/\sqrt{n})D$$

where n is the number of nonmissing values. This value is used to interpolate linearly within the range of simulated critical values given in Stephens (1974).

Plots

When the PLOT option is specified in the PROC UNIVARIATE statement, the procedure generates three data plots.

The first plot is a stem-and-leaf plot (Tukey 1977) if no more than forty-eight observations fall into a single interval. Otherwise, the procedure plots a horizontal bar chart.

The second plot is a box plot or schematic plot. The bottom and top edges of the box are located at the sample 25th and 75th percentiles. The center hori-

zontal line is drawn at the sample median and the central plus sign (+) is at the sample mean. It is possible for all of these statistics to fall on the same printer line. The central vertical lines, called "whiskers," extend from the box as far as the data extend, to a distance of at most 1.5 interquartile ranges. (An interquartile range is the distance between the 25th and the 75th sample percentiles.) Any value more extreme than this is marked with a zero if it is within three interquartile ranges of the box, or with an asterisk (*) if it is still more extreme. For more explanation about this plot, see Tukey (1977).

The third plot, a normal probability plot, is a quantile-quantile plot of the data. The empirical quantiles are plotted against the quantiles of a standard normal distribution. Asterisks (*) mark the data values. The vertical coordinate is the data value, and the horizontal coordinate is

$$\Phi^{-1}((r_i - 3/8)/(n + 1/4))$$

where r_i is the rank of the data value, Φ^{-1} is the inverse of the standard normal distribution function, and n is the number of nonmissing data values. The plus signs (+) provide a reference straight line that is drawn using the sample mean and standard deviation. If the data are from a normal distribution, they should tend to fall along the reference line.

Computer Resources

The data are stored internally in single precision. For most efficient processing, an amount of temporary storage of $20\Sigma_i n_i$ bytes is required where n_i is the number of unique values for the ith variable. The minimum amount of temporary storage required is 20 MAX (n_i) in bytes. Additional storage is required for buffers for each output data set and for the input data set.

Printed Output

For each variable, PROC UNIVARIATE prints:

1. VARIABLE=, the name of the variable
2. the variable label
3. N, the number of observations on which the calculations are based
4. SUM WGTS, the sum of the weights of these observations
5. the MEAN
6. the SUM
7. STD DEV, the standard deviation
8. the VARIANCE
9. the measure of SKEWNESS
10. the measure of KURTOSIS
11. USS, the uncorrected sum of squares
12. CSS, the corrected sum of squares
13. CV, the coefficient of variation
14. STD MEAN, the standard error of the mean
15. T: MEAN=0, the Student's t value for testing the hypothesis that the population mean is 0
16. PROB> |T|, the probability of a greater absolute value for this t value
17. SGN RANK, the centered signed rank statistic for testing the hypothesis that the population mean is 0
18. PROB> |S|, an approximation of the probability of a greater absolute value for this statistic under the hypothesis that the population mean is 0
19. NUM¬=0, the number of nonzero observations

20. MAX, the largest value
21. Q3, Q1, and MED, the upper and lower quartiles, and the median
22. MIN, the smallest value
23. the RANGE
24. Q3-Q1, the difference between the upper and lower quartiles
25. the MODE
26. the 1st, 5th, 10th, 90th, 95th, and 99th percentiles
27. the five largest, HIGHEST, and five smallest, LOWEST, values.

If missing values occur for a variable, the following are printed:

28. the MISSING VALUE
29. COUNT, the number of occurrences
30. %COUNT/NOBS, the count as a percentage of the total number of observations
31. %COUNT/NMV, the count as a percentage of the total number of missing values (not shown).

UNIVARIATE can also print:

32. W:NORMAL...D:NORMAL, test statistics
33. associated probabilities, PROB<W or PROB>D, for testing the hypothesis that the data come from a normal distribution
34. STEM LEAF, a stem-and-leaf plot (if any value's count is greater than forty-eight, a horizontal bar chart is printed instead)
35. BOXPLOT, a box plot
36. a NORMAL PROBABILITY PLOT
37. VALUE, a frequency table of variable values
38. COUNT, frequencies
39. CELL, percentages
40. CUM, cumulative percentages.

EXAMPLES

State Data: Example 1

In the following example, descriptive statistics are produced for the 1980 population of each of the fifty states (plus two fictional states to illustrate missing values). The ID statement makes it possible to identify the states with extreme populations.

```
DATA STATEPOP;
   INPUT STATE $ POP @@;
   LABEL POP = '1980 CENSUS POPULATION IN MILLIONS';
   CARDS;
ALA  3.90 ALASKA 0.40 ARIZ   2.72 ARK  2.29 CALIF 23.67
COLO 2.89 CONN   3.11 DEL    0.59 FLA  9.75 GA     5.46
HAW  0.96 IDAHO  0.94 ILL   11.43 IND  5.49 IOWA   2.91
KAN  2.36 KY     3.66 LA     4.20 ME   1.12 MD     4.22
MASS 5.74 MICH   9.26 MINN   4.07 MISS 2.52 MO     4.95
MONT 0.79 NEB    1.57 NEV    0.80 NH   0.92 NJ     7.36
NM   1.30 NY    17.56 NC     5.88 ND   0.65 OHIO  10.80
OKLA 3.02 ORE    2.63 PA    11.86 RI   0.95 SC     3.12
SD   0.69 TENN   4.59 TEXAS 14.23 UTAH 1.46 VT     0.51
VA   5.35 WASH   4.13 W.VA   1.95 WIS  4.71 WYO    0.47
```

```
YY     .   ZZ       .
;
PROC UNIVARIATE FREQ PLOT NORMAL;
   VAR POP;
   ID STATE;
```

Ouptut 54.1 Descriptive Statistics for 1980 Population: PROC UNIVARIATE

```
                                                                                        1
                                          UNIVARIATE
(1)                  (2)
VARIABLE=POP         1980 CENSUS POPULATION IN MILLIONS

              MOMENTS                                                  QUANTILES(DEF=4)

N (3)              50  SUM WGTS (4)     50          (20) 100% MAX     23.67     99%      23.67
MEAN (5)       4.5182  SUM (6)      225.91               75% Q3      5.4675     95%    15.7285
STD DEV (7)   4.71588  VARIANCE (8) 22.2395         (21) 50% MED      3.065     90%     11.367
SKEWNESS (9)  2.13788  KURTOSIS (10) 5.36667             25% Q1        1.08 (26) 10%   0.654
USS (11)      2110.44  CSS (12)      1089.74        (22)  0% MIN        0.4      5%      0.492
CV (13)       104.375  STD MEAN (14) 0.666926                                    1%        0.4
T:MEAN=0 (15) 6.77467  PROB>|T| (16)  0.0001        (23) RANGE        23.27
SGN RANK (17)   637.5  PROB>|S| (18)  0.0001        (24) Q3-Q1       4.3875
NUM ^= 0 (19)      50                               (25) MODE           0.4
W:NORMAL (32) 0.771308 PROB<W (33)       <.01

                                          EXTREMES

                              (27) LOWEST    ID          HIGHEST    ID
                                     0.4(ALASKA  )       11.43(ILL      )
                                    0.47(WYO     )       11.86(PA       )
                                    0.51(VT      )       14.23(TEXAS    )
                                    0.59(DEL     )       17.56(NY       )
                                    0.65(ND      )       23.67(CALIF    )

                              (28) MISSING VALUE          .
                                   (29)     COUNT          2
                              (30)  % COUNT/NOBS        3.85

STEM LEAF (34)                  #     (35) BOXPLOT           (36) NORMAL PROBABILITY PLOT
  22 7                          1          *           23+                                        *
  20                                                     |
  18                                                     |
  16 6                          1          0             |                                     *     +
  14 2                          1          0             |                                   *   ++++
  12                                                     |                                   +++++
  10 849                        3          |             |                               **+*
   8 38                         2          |             |                           ++**
   6 4                          1          |             |                       ++++ *
   4 112267935579              12       +--+--+          |                   +++*********
   2 345679901179              12       *-----*          |               +*******
   0 455667889990135 69        17       +-----+         1+   *    * ** **********
     ----+----+----+----+                                 +----+----+----+----+----+----+----+----+----+----+
                                                               -2        -1         0        +1        +2
```

```
                                             SAS                                                2
                                          UNIVARIATE
VARIABLE=POP         1980 CENSUS POPULATION IN MILLIONS

                                       FREQUENCY TABLE
         (37)  (38)   (39)  (40)
                      PERCENTS                         PERCENTS                          PERCENTS
         VALUE COUNT  CELL   CUM    VALUE COUNT   CELL   CUM     VALUE COUNT   CELL   CUM
           0.4     1   2.0   2.0     2.29     1    2.0  36.0      4.71     1    2.0  70.0
          0.47     1   2.0   4.0     2.36     1    2.0  38.0      4.95     1    2.0  72.0
          0.51     1   2.0   6.0     2.52     1    2.0  40.0      5.35     1    2.0  74.0
          0.59     1   2.0   8.0     2.63     1    2.0  42.0      5.46     1    2.0  76.0
          0.65     1   2.0  10.0     2.72     1    2.0  44.0      5.49     1    2.0  78.0
          0.69     1   2.0  12.0     2.89     1    2.0  46.0      5.74     1    2.0  80.0
          0.79     1   2.0  14.0     2.91     1    2.0  48.0      5.88     1    2.0  82.0
           0.8     1   2.0  16.0     3.02     1    2.0  50.0      7.36     1    2.0  84.0
```

(continued on next page)

(continued from previous page)

0.92	1	2.0	18.0	3.11	1	2.0	52.0	9.26	1	2.0	86.0			
0.94	1	2.0	20.0	3.12	1	2.0	54.0	9.75	1	2.0	88.0			
0.95	1	2.0	22.0	3.66	1	2.0	56.0	10.8	1	2.0	90.0			
0.96	1	2.0	24.0	3.9	1	2.0	58.0	11.43	1	2.0	92.0			
1.12	1	2.0	26.0	4.07	1	2.0	60.0	11.86	1	2.0	94.0			
1.3	1	2.0	28.0	4.13	1	2.0	62.0	14.23	1	2.0	96.0			
1.46	1	2.0	30.0	4.2	1	2.0	64.0	17.56	1	2.0	98.0			
1.57	1	2.0	32.0	4.22	1	2.0	66.0	23.67	1	2.0	100.0			
1.95	1	2.0	34.0	4.59	1	2.0	68.0							

Census Data: Example 2

See "PROC Step Applications" for another example using PROC UNIVARIATE to analyze selected characteristics of U.S. cities with populations over 50,000 according to *County and City Data Book, 1983*. The approaches of the SUMMARY and TABULATE procedures to these same data are compared.

REFERENCES

Data User Services Division (1983), *County and City Data Book*, Files on Tape Technical Documentation, Washington: Bureau of the Census.

Fisher, R.A. (1973), *Statistical Methods for Research Workers*, 14th Edition, New York: Hafner Publishing Company.

Lehmann, E.L. (1975), *Nonparametrics: Statistical Methods Based on Ranks*, San Francisco: Holden-Day, Inc.

Shapiro, S.S. and Wilk, M.B. (1965), "An Analysis of Variance Test for Normality (complete samples)," *Biometrika*, 52, 591-611.

Stephens, M.A. (1974), "EDF Statistics for Goodness of Fit and Some Comparisons," *Journal of the American Statistical Association*, 69, 730-737.

Tukey, J.W. (1977), *Exploratory Data Analysis*, Reading, Massachusetts: Addison-Wesley.

Chapter 55

The XCOPY Procedure

Operating systems: CMS, OS, and VSE

ABSTRACT

The XCOPY procedure performs two operations on SAS data sets. XCOPY converts SAS data sets to transport format from the standard format of the operating system under which they were created. A SAS data set in transport format can be read by the SAS System running under the AOS/VS, PRIMOS, or VMS operating system. XCOPY also converts SAS data sets in transport format to standard format. XCOPY reads SAS data sets in transport format (created under AOS/VS, PRIMOS, or VMS) and converts them to standard CMS, OS, or VSE format.

INTRODUCTION

You may want to refer to chapters on the DATA step and PROC step to familiarize yourself with terminology used in this procedure description. Also, refer to the "SAS Files" chapter for a discussion of SAS data sets, transport-format data sets, and the differences between the two.

In this discussion, *standard format* refers to the organization of a SAS data set created in a CMS, OS, or VSE environment. *Transport format* is a format suitable

for transporting to or from an AOS/VS, PRIMOS, or VMS operating system. A SAS data set in transport format cannot be processed by SAS procedures or used as input in a DATA step. It must be converted to standard format with XCOPY.

The XCOPY procedure:

- converts a SAS data set in transport format to a SAS data set in the standard format for your operating system.
- converts a SAS data set in standard format to a SAS data set in transport format.

PROC XCOPY performs these operations on SAS data sets only. If the SAS data library contains other types of files, XCOPY ignores them. (See the "SAS Files" chapter for information on the various types of files, other than SAS data sets, that a SAS data library can contain.) XCOPY reformats all the SAS data sets in the SAS data library during one XCOPY operation, or you can select the data sets to be reformatted from the library.[1]

When you change one or more SAS data sets into transport format from the format of the operating system under which they were created, you create a *transport data library*. Use the XCOPY procedure to create a transport data library that can be moved to a SAS System running under AOS/VS, PRIMOS, or VMS. Also, use the XCOPY procedure to read a transport data library created by a SAS System running under AOS/VS, PRIMOS, or VMS and convert it to standard format.

XCOPY is not suitable for transporting SAS data sets containing non-standard characters, such as those produced by non-standard hexidecimal literals.

XCOPY vs. COPY

You can use PROC XCOPY to read and write transport data libraries only under CMS, OS, and VSE operating systems. The XCOPY procedure is not available with the SAS System running under AOS/VS, PRIMOS, and VMS; in these environments use the COPY procedure with IMPORT and EXPORT options to read and write transport data libraries. The IMPORT and EXPORT options are ignored on the PROC COPY statement under CMS, OS, and VSE.

SPECIFICATIONS

PROC XCOPY IN=*libref* OUT=*libref* *option*;
  SELECT *SASdataset* ...;
  EXCLUDE *SASdataset* ...;

PROC XCOPY Statement

PROC XCOPY IN=*libref* OUT=*libref* EXPORT | IMPORT;

Both the IN= and OUT= options must be specified, and either the EXPORT or IMPORT option must be specified.

IN=*libref*
: specifies the SAS data library to be used for input. To define the library, associate the *libref* with the name of the library in the control language for your job.

OUT=*libref*
: specifies the SAS data library to be used for output. To define the library, associate the *libref* with the name of the library in the control language for your job.

EXPORT
specifies that the input library (IN=) is a SAS data library in standard format that is to be converted to a transport data library.

IMPORT
specifies that the input library (IN=) is a transport data library that is to be converted to standard format.

SELECT Statement

SELECT *SASdataset ...;*

In the SELECT statement you specify the names of one or more SAS data sets in the input library that you want to convert and then write to the output library. All selected data sets must belong to the library referenced by the IN= option.

There are several ways to specify the SELECT statement. You can select a single data set to be converted with the statements:

```
PROC XCOPY IN=TRANS OUT=PORT EXPORT;
   SELECT TEST;
```

You can select more than one data set by listing the name of each:

```
PROC XCOPY IN=TRANS OUT=PORT EXPORT;
   SELECT TEST TEST2;
```

You can select a range of data sets. The statements

```
PROC XCOPY IN=TRANS OUT=PORT EXPORT;
   SELECT DATA1-DATA3;
```

select DATA1, DATA2, and DATA3.

Specify one or more characters followed by a colon (:) to select a group of data sets having names beginning with those characters. For example, the statements

```
PROC XCOPY IN=TRANS OUT=PORT EXPORT;
   SELECT TEST AB:;
```

convert the SAS data set TEST and all SAS data sets having names that begin with AB.

In all of the above examples, data sets are selected from a library referenced by the libref TRANS, converted, and written to the library referenced by libref PORT.

EXCLUDE Statement

EXCLUDE *SASdataset ...;*

Use the EXCLUDE statement to specify the names of one or more SAS data sets that you want to exclude from the output library. The data sets named in the EXCLUDE statement are **not** converted. You can specify SAS data set names in the EXCLUDE statement using the same syntax shown for the SELECT statement examples. As shown in the following example, you can specify single data set names, a range of data sets, and a group of SAS data sets:

```
PROC XCOPY IN=PORT OUT=SAVELIB IMPORT;
   EXCLUDE DS1 STAT4-STATS6 XYZ:;
```

When this program executes, all data sets in the transport data library referenced by PORT are converted to standard format, **except** DS1, STAT4, STAT5, STAT6, and data sets beginning with the letters XYZ.

Note: you cannot use both a SELECT and an EXCLUDE statement in the same PROC step.

DETAILS

Input data set If you want to change one or more SAS data sets to transport format, you are *exporting* a transport data library. The libref specified with the IN= option refers to the SAS data library in standard CMS, OS, or VSE format, and you must specify the EXPORT option.

If you are changing a transport data library to a SAS data library in standard format, you are *importing* a data library. The libref specified with the IN= option refers to a transport data library, and you must specify the IMPORT option.

The input library can be located on either disk or tape.

Output data set If you want to change one or more SAS data sets to transport format, you are *exporting* a transport data library. The libref specified with the OUT= option refers to a transport data library, and you must specify the EXPORT option.

If you are changing a transport data library to standard format, you are *importing* a data library. The libref specified with the OUT= option refers to a SAS data library in standard format, and you must specify the IMPORT option.

An output data set can be located on either disk or tape. Unless you are sending it across a network, however, the output data set must be transferred to a tape before it can be read by the AOS/VS, PRIMOS, or VMS operating system.

Printed output A message is written on the SAS log for each SAS data set copied. No other output is produced.

EXAMPLES

Using PROC XCOPY under OS: Example 1

General information on OS control language for SAS jobs can be found in the *SAS Companion for OS Operating Systems and TSO*. You must define both libraries used by the XCOPY procedure in the control language used to invoke SAS. If you invoke SAS for batch execution, use two DD statements: one to define the transport library and another to define the standard format library. The DDnames specified in the respective DD statements are then used as librefs for the IN= and OUT= options in the PROC statement, as shown in the following examples.

Note: you can also use XCOPY under TSO (the interactive monitor of the OS operating system). However, most installations prohibit or limit the use of tape files under TSO. If your XCOPY application involves the use of tape files, see your systems personnel or SAS consultant for information on executing XCOPY interactively under TSO.

Creating a transport data library The following example shows the OS batch control language and SAS statements necessary to create a transport data library on disk:

```
// JOB accountinginformation
// EXEC SAS
//STD1 DD DSN=OSFORMAT.SAS.DATALIB,DISP=OLD
//TRANS1 DD DSN=TRANS.FORMAT.LIBRARY,DISP=(NEW,KEEP),
//   UNIT=DISK,DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000),
//   SPACE=(TRK,(15,0)),VOL=SER=ABC123
//SYSIN DD *
PROC XCOPY IN=STD1 OUT=TRANS1 EXPORT;
/*
```

In this job the first DD statement defines the SAS data library in OS standard format to be converted to transport format. STD1 is then used as the libref for the IN= option in the PROC statement. The second DD statement defines the transport data library to be created and stored on disk. TRANS1 is then used as the libref for the OUT= option in the PROC statement.

Once created, the transport data library can be sent across a network (if one exists) to a machine that runs another operating system supported by the SAS System. Consult your SAS consultant or systems personnel for information on using the network. If you will not be using a network, you will probably want to store the transport data library on tape.

A tape containing a transport data library must be nonlabeled and have a density compatible with the maximum density of the destination machine. You should write only one transport library on each tape; however, the library can contain as many members as you want.

Reading a transport data library The following example uses XCOPY to read a transport data library from tape and store it in a disk data set:

```
//  JOB  accountinginformation
//  EXEC  SAS
//TRANS2  DD DSN=TRANSPT.FORMAT.LIBRARY,DISP=OLD,
//  UNIT=TAPE,VOL=SER=XYZ345,LABEL=(1,NL),
//  DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000,DEN=n)
//STD2  DD  DSN=OSFORMAT.SAS.DATALIB,
//  DISP=(,KEEP),UNIT=DISK,
//  DCB=DSORG=DA,SPACE=(TRK,(15,0)),VOL=SER=ABC123
//SYSIN  DD *
PROC XCOPY IN=TRANS2 OUT=STD2  IMPORT;
/*
```

In this job the first DD statement defines the transport data library to be IMPORTed, and TRANS2 is specified in the IN= option. The second DD statement defines the OS-format SAS data library that is created and stored on disk, and STD2 is specified in the OUT= option.

If you want to create a transport data library on tape, be sure to include DCB information because the new data set is a transport format library, not a tape format library. Since TAPExxxx is a special libref reserved for SAS data sets written in tape format, you should not use librefs beginning with "TAPE" when reading or creating transport data libraries.

Using PROC XCOPY under CMS: Example 2

Commands required when using XCOPY under CMS are described in this section. For general information on commands needed for SAS jobs under CMS, see the *SAS Companion for the VM/CMS Operating System*.

The transport data library must be defined by a FILEDEF command before you invoke PROC XCOPY. When you define the transport library, you must specify record and blocking attributes. Use the DDname specified in the FILEDEF as the libref in the IN= or OUT= option, as appropriate.

The SAS data library in standard format need not be defined by a FILEDEF unless you have created SAS data sets with identical names on different minidisks and you want to access data sets on a minidisk that is not first in the search order.

When you convert one or more SAS data sets to transport format, all of the transport-format data sets are written to a transport data library as a single file. When the transport data library is converted back to standard format, the SAS data sets are separated into individual files again. See the example below.

Transport data libraries on disk The following example uses XCOPY to create a transport data library on disk. Assume that there are three SAS data sets in the library referenced by MYSAS:

```
R;
FILEDEF  PORT  DISK  NEWFILE  OUT  B (RECFM FB LRECL 80 BLKSIZE 8000)
R;
SAS
SAS messages
   1?
PROC  XCOPY  IN=MYSAS  OUT=PORT  EXPORT;
   2?
RUN;
```

When this program executes, XCOPY reads three SAS data sets from the library referenced by MYSAS (IN=), converts them to transport format, and creates the transport data library referenced by PORT (OUT=).

Once created, the transport data library can be sent across a network to be used with the SAS System running under another operating system. Otherwise, the transport library must be written to tape. If your installation supports a network between operating systems, you can obtain information on using the network from the SAS consultant or systems personnel at your site.

Transport data libraries on tape A tape on which a transport data library is written must be nonlabeled and have a density compatible with the maximum density of the destination machine. You should write only one transport library on each tape; however, the transport library can contain as many members as you want. The following is a typical FILEDEF for writing a transport library on tape:

```
FILEDEF libref TAPn NL (RECFM FB LRECL 80 BLKSIZE 8000 DEN n)
```

Since TAPExxxx is a special libref reserved for SAS data sets written in tape format, you should not use librefs beginning with "TAPE" for transport format data sets.

Using PROC XCOPY under VSE: Example 3

General information on VSE JCL for SAS jobs is available in the *SAS Companion for the VSE Operating System*. This section discusses the control language needed to define files that are to be imported or exported using XCOPY.

Creating a transport data library The following example uses PROC XCOPY to read a SAS library in VSE format, convert it, and write a transport data library to tape:

```
* $$ JNM=JOBNAME ...
* $$ LST LST= ...
// JOB accountinginformation
// PAUSE PLEASE MOUNT TAPE VOL CWM123 // ASSGN SYS222 TO IT
// MTC REW,SYS222
// UPSI XXXX1
// DLBL STD,'VSEFORM.SAS.DATALIB'
// EXTENT SYS050,XYZ123
// ASSGN SYS050,DISK,VOL=XYZ123,SHR
// EXEC  SASVSE,SIZE=64K
PROC XCOPY IN=STD OUT=OXXX222 EXPORT;
/*
/&
* $$ EOJ
```

The DLBL statement with filename STD defines the SAS data library to be EXPORTed. STD is then used as the libref for the IN= option in the PROC XCOPY statement. The libref OXXX222 specified for the OUT= option is associated with the tape logical unit SYS222 to define the transport data library to be created and stored on tape.

A tape on which a transport data library is written must be nonlabeled and be recorded at a density that can be read on the destination machine. You should write only one transport library on each tape; however, the transport library can contain as many members as you want.

Reading a transport data library This example uses XCOPY to read a transport data library from tape, convert it to standard VSE format, and store it on disk:

```
* $$ JNM=JOBNAME ...
* $$ LST LST= ...
// JOB  accountinginformation
// PAUSE PLEASE MOUNT TAPE VOL CWM010 // ASSGN SYS222 TO IT
// MTC REW,SYS222
// MTC FSF,SYS222
// UPSI XXX1
// DLBL ASTD,'VSEFORM.SAS.DATALIB',99/365
// EXTENT SYS111,ABC321,1,0,800,15
// ASSGN SYS111,DISK,VOL=ABC321,SHR
// EXEC SASVSE,SIZE=64K
PROC XCOPY IN=XXX222 OUT=ASTD IMPORT;
/*
/&
* $$ EOJ
```

NOTE

1. **CMS**: A SAS data library is a logical concept, not a physical entity. All SAS files with the same filetype and filemode belong to the same logical SAS data library.

   OS and VSE: A SAS data library is a physical file—an OS or VSE file that contains one or more SAS files.

APPENDICES

Version 5 Changes and Enhancements to Base SAS® Software

Operating System Notes

Full-Screen Editing

Appendix 1

Version 5 Changes and Enhancements to Base SAS® Software

INTRODUCTION

This appendix contains three sections:

- changes between Version 4 on minicomputers and Version 5
- changes between the 82.4 release on mainframes and Version 5
- operating system differences.

The sections that apply to you depend on the operating systems you were using, those you are currently using, and any previous versions of the SAS System you were using.

CHANGES BETWEEN VERSION 4 AND VERSION 5

Version 4 of the SAS System was released for the AOS/VS, PRIMOS, and VMS operating systems as documented in SAS Technical Reports P-128, "Changes and Enhancements in the Base SAS and SAS/GRAPH Products Under VMS, Version 4"; P-129, "Changes and Enhancements in the Base SAS and SAS/GRAPH Products under AOS/VS, Version 4"; and P-130, "Changes and Enhancements in the Base SAS and SAS/GRAPH Products under PRIMOS, Version 4."

The following changes have been made between Version 4 and Version 5:

Statements Used in the DATA Step

ARRAY allows multidimensional explicitly subscripted arrays.

SAS Statements Used Anywhere

FILENAME associates a fileref (a reference name) with an external file.

SAS System Options

DMS/NODMS controls whether the SAS Display Manager System is available.

In addition, several new options applicable to SAS/GRAPH software are available, and many option descriptions have changed.

CHANGES BETWEEN 82.4 RELEASE AND VERSION 5

Release 82.4 was available for the CMS, OS, and VSE operating systems as documented in the *SAS User's Guide: Basics, 1982 Edition* and *SAS User's Guide: Statistics, 1982 Edition*.

The following changes have been made between the 82.4 release and Version 5.

SAS Language

Numbered variable lists can be specified in ascending or descending order, as in X1-X5 or X5-X1.

Many statements and options now require that character strings be enclosed in single or double quotes.

The variable list words CHARACTER and _CHARACTER_ can be abbreviated CHAR and _CHAR_.

Statements Used in the DATA Step

ARRAY allows explicitly subscripted arrays.

ATTRIB allows you to specify the format, informat, label, and length of a SAS variable in a single statement.

DO WHILE/UNTIL clause is available in iterative DO statement.

END now ends SELECT groups as well as DO groups.

INPUT can include explicitly subscripted array references; commas can be used between items in format lists to delineate separate informats.

PUT allows commas in format lists to distinguish between formats; PUT _INFILE_ is available for VSAM files; specifications can include explicitly subscripted array references.

SELECT selects one of several statements to be executed.

SAS Functions

ERFC is the complement to the ERF function.

DIM returns number of elements in array.

The PROC Step

Prefix variable lists are available in most procedures. Specifying the first letters of a variable name followed by a colon (:) refers to all variables beginning with those letters. For example, MFK: means all variables whose names begin with the letters MFK.

Statements Used in the PROC Step

ATTRIB allows you to specify the format, informat, label, and length of one or more SAS variables in a single statement.

DROP is not available in the PROC step; use the DROP= data set option instead.

KEEP is not available in the PROC step; use the KEEP= data set option instead.

SAS Statements Used Anywhere

FOOTNOTE prints footnotes at the bottom of the page; text must be enclosed in quotes.

HELP additional options are available.

%PUT automatically invokes the %PUT statement from the SAS macro language, unless the NOMACRO option is specified.

RUN CANCEL or QUIT option ends a DATA or PROC step without executing it.

TITLE text must be enclosed in quotes.

SAS Display Manager

The SAS Display Manager System is a full-screen facility with an editor that allows you to interact with all parts of your SAS job.

SAS Files

SAS data libraries can now contain other types of SAS files in addition to SAS data sets, including catalogs for full-screen procedures, graphics catalogs, user-written formats, SAS/ETS models, and SAS/IML saved workspaces.

SAS System Options

DMS/NODMS
controls whether the SAS Display Manager System is available.

IMPLMAC/NOIMPLMAC
: controls whether macros defined as statement-style macros can be invoked with statement-style macro calls.

MPRINT/NOMPRINT
: controls printing of SAS macro statements generated by macro execution.

PROBSIG=
: controls formatting of p-values in statistical procedures.

In addition, several new options applicable to SAS/GRAPH software and new options for reading VSAM files have been added, and many option descriptions have changed.

SAS Macro Language

Statements

%INPUT	accepts user input in conversational environment.
%MACRO	uses the STMT option to define statement-style macros.
%PUT	writes text to SAS log; now part of macro facility.

Functions

%BQUOTE	removes meaning from unanticipated special characters (except %, & and mnemonic operators) during macro execution.
%NRBQUOTE	removes meaning from all unanticipated special characters during macro execution.
%NRQUOTE	removes meaning from special characters including % and & during macro execution.
%NRSTR	removes meaning from special characters including % and & at macro compilation.
%QUOTE	removes meaning from special characters except % and & during macro execution.
%UNQUOTE	restores recognition of special characters.

Automatic macro variables

SYSBUFFER	receives text entered in response to %INPUT statement.

SAS Procedures

APPEND	BASE= data set becomes current SAS data set.
BROWSE	see EDITOR procedure changes.
CALENDAR	can include holidays; allows formats for sum and mean variables; provides new values for FORMCHAR= option.
COMPARE	is a new procedure that compares values of variables in two SAS data sets and reports differences.
CONTENTS	allows you to list contents of all types of SAS files.
COPY	permits SELECT and EXCLUDE statements in the same PROC step; allows you to copy all types of SAS files within an operating environment.

DATASETS	has a full-screen version and can perform some operations on new types of SAS files.
DELETE	is no longer in the base SAS System.
EDITOR	no longer requires a RUN statement; allows you to assign variable attributes anywhere within the session; allows global statements such as TITLE, FOOTNOTE, and OPTIONS anywhere within the session; has LIST option in VERIFY command.
FORMAT	HIGH, LOW, OTHER options can be used for character formats; noninclusive range can be specified.
MEANS	SUMWGT option and WEIGHT statement added.
PDSCOPY	contains option to suppress how aliases are to be matched; allows you to specify maximum blocksize of text records.
SUMMARY	allows printing of variables in descending order and specifying order of class variable; has an option to specify denominator in computation of weight variance.
TABULATE	interleaves crossed subgroups; uses SAS or user-defined formats in table cells; uses analysis variables in denominator definitions; CONDENSE option puts multiple logical pages on a single physical page; BOX option puts page dimension text, variable label, or character string in box over row titles.
TRANSPOSE	provides options for outputting variables to SAS data set.

OPERATING SYSTEM DIFFERENCES

A few features of the base SAS software product work differently or are supported only under specific operating systems. The following table indicates the major features that are different under the operating systems specified. You need to use this table if you are preparing a SAS program to be used under more than one operating system or if you want to execute programs on one operating system that were developed under another operating system.

Table A1.1 Operating System Differences

Feature	Operating System					
	AOS/VS	CMS VM/PC	OS	PRIMOS	VMS	VSE
SAS Language						
Minimum length of numeric variable is 3.				X		
Statements Used in the DATA Step						
ARRAY statement: all arrays, including multidimensional arrays, are explicitly subscripted.	X			X	X	
ARRAY statement: explicitly subscripted one-dimensional arrays are available; use nested implicitly subscripted arrays for effect of multidimensional array processing.		X	X			X
ATTRIB statement: allows more than one variable name.	X			X	X	
BY statement: if used, must follow SET, MERGE, or UPDATE statement.	X			X	X	
CALL statement: allows SAS and user-written routines, including VNAME and LABEL.		X	X			X
CALL statement: used only to invoke SAS random number subroutines.	X			X	X	
CARDS and CARDS4 statements: cannot have statement label.	X			X	X	
DO OVER statement: not supported.	X			X	X	
ERROR statement: not supported.	X			X	X	

continued on next page

Table A1.1 *Continued*

Feature	Operating System					
	AOS/VS	CMS VM/PC	OS	PRIMOS	VMS	VSE
FILE statement: now writes to files that have a special access method, such as VSAM files and VSE direct access files; provides options to control action when SAS attempts to write past the end of the current line.		X	X			X
INFILE statement: has options for reading VSAM files.		X	X			X
INFILE statement: has options for reading DA and SD files and POWER queue entries.						X
INFILE statement: has options for reading VTOC files.			X			X
INPUT statement: n* format modifier cannot be used; named input not supported; decimal point must precede decimal specifications following column input.	X			X	X	
PUT statement: n* modifier not supported; a decimal point must precede the decimal format specification in column output style; format can be followed by alignment specification.	X			X	X	
SELECT statement: SELECT expression is optional.		X	X			X

continued on next page

Table A1.1 *Continued*

Feature	Operating System					
	AOS/VS	CMS VM/PC	OS	PRIMOS	VMS	VSE
SAS Expressions						
Redundant comparison operators not allowed: NG, NL, NOT=, =<, =>.	X			X	X	
Caret symbol (^) is used for NOT operator.	X			X	X	
Statements Used in the PROC Step						
PARMCARDS and PARMCARDS4 statements are available.		X	X			
SAS Statements Used Anywhere						
CLEAR statement: is not supported.	X			X	X	
COMMENT keyword: is not supported; other types of comments are available.	X			X	X	
FILENAME statement: associates SAS fileref (reference name) with external file.	X			X	X	
%INCLUDE statement: options are not supported.	X			X	X	
LIBNAME statement: defines libref (reference name) for directory.	X			X	X	
LIBSEARCH statement: specifies list of librefs that identify directories to search for input SAS data sets and user-defined formats.	X			X	X	

continued on next page

Table A1.1 *Continued*

Feature	Operating System					
	AOS/VS	CMS VM/PC	OS	PRIMOS	VMS	VSE
X statement: target operating system can be specified.	X			X	X	
X statement: quotes around command are optional.		X	X			X
SAS Informats and Formats						
Decimal range is 0-31 for informats w.d, BZw.d, COMMAw.d, ZBw.d, PDTIME.		X	X			X
PDTIME., RMFDUR., RMFSTAMP., SMFSTAMP., TODSTAMP., TU., and column-binary informats are not available.	X			X	X	
SAS Files						
Data set options BLKSIZE=, BUFNO=, FILECLOSE=, FILEDISP=, GEN=, PROTECT=, READ=, and REPLACE= are not available.	X			X	X	
Data set option TRANSPORT= is not available.		X	X			X
SAS System Options						
SAS system options vary by operating system. Some new options are available. See chapter for table.	X	X	X	X	X	X

continued on next page

Table A1.1 *Continued*

Feature	Operating System					
	AOS/VS	CMS VM/PC	OS	PRIMOS	VMS	VSE
SAS Procedures						
APPEND cannot append BASE= data set to itself.	X			X	X	
CONTENTS options HISTORY and SOURCE are not available.	X			X	X	
CONVERT procedure is not supported.	X			X	X	
COPY reformats a SAS data set to produce a transport data set and vice versa.	X			X	X	
FORMAT procedure PRINT and TEXT options are ignored.	X				X	
FORMAT stores formats as external files.		X	X			X
FORMAT stores formats as SAS files.	X			X	X	
SOURCE procedure processes VSE source statement sublibraries.						X
SUGI Supplemental Library is available.		X	X			X
XCOPY reformats a SAS data set to produce a transport data set and vice versa.		X	X			X

Appendix 2

Operating System Notes

INTRODUCTION

This appendix provides a general description of the basic functions of an operating system and how the operating system and the SAS System interact to execute your SAS program. You will also learn about SAS features that relate to file access.

A detailed section for each operating system includes examples to illustrate these features.

To use SAS software products, you do not need to be an expert on operating systems; however, as you learn more about your operating system, you will be able to do more with SAS. After reading this material, you should have a better idea of how the operating system and the SAS System work together and where to find more information.

THE OPERATING SYSTEM

Your SAS programs execute in an environment controlled by a set of powerful programs called an *operating system*. The operating system controls all work done by the computer, such as allocating computer resources to run programs and storing data. Because an operating system oversees activities of the computer, it is sometimes called the *host environment*.

There are different operating systems for different types of computers. However, all operating systems are designed to handle certain basic tasks:

- accept jobs for execution
- store and retrieve data files
- manage terminal sessions
- allocate resources, such as internal memory, time, and disk space, to individual jobs
- control the action of peripheral equipment, such as printers, plotters, and disk and tape drives.

In the same way that you communicate with the SAS System using the SAS language, you communicate with an operating system using the operating system's language. We refer to operating system languages generally as *control languages*. **Table A2.1** lists the types of computers and operating systems under which the SAS System runs, and it gives the commonly used names of their control languages.

INTERACTIONS BETWEEN THE OPERATING SYSTEM AND THE SAS USER

Each time a SAS program executes, many interactions between the SAS System and the operating system occur. The amount of interaction that you need to be aware of depends on your program.

Consider the following example:

```
DATA WEATHER;
   INPUT TEMP PRECIP SUNHRS;
   CARDS;
85.5 0.00 12.33
84.2 0.09 12.31
79.5 0.10 12.30
;
PROC PRINT DATA=WEATHER;
   TITLE 'WEATHER INFORMATION';
```

Table A2.1 Computers, Operating Systems, and Control Languages

Computers	Operating Systems	Control Language Name
Data General ECLIPSE MV	AOS/VS	Command Line interpreter (CLI)
Digital VAX	VMS	Digital Command Language (DCL)
IBM and compatible mainframes*	OS	Job Control Language (JCL)
	TSO (OS interactive)	TSO Command Language
	CMS	CMS and CP Command Language
	VM/PC	CMS and CP Command Language
	VSE	Job Control Language (JCL)
	ICCF (VSE interactive)	ICCF Command Language
PRIME	PRIMOS	Command Procedure Language (CPL)

*The 370, 308X, and 4300 series are examples of IBM mainframes. Amdahl 470 V6 is an example of an IBM-compatible mainframe.

Even in the simplest SAS execution, like this one, you need to be aware of two interactions:

1. *Invoking SAS.* You make a request to the operating system to use the SAS software.
2. *Receiving output.* The output from a SAS program is printed, displayed on a terminal, or written to a file.

Many of the SAS programs you write may involve no more interaction than our example. However, you need to be aware of two fairly common situations in SAS programming that require you to specify information to the operating system:

1. *Defining permanent files.* You can define a file when you want to read or write data that are stored independently from your SAS program.
2. *Rerouting output.* You can send output from a SAS program to the destination of your choice rather than to its default destination.

The following pages discuss each of these common situations in more detail.

Invoking the SAS System

In order to run a SAS program, you must issue a request to the operating system to use the SAS System. The request is made using the operating system's control language, for example, with the SAS command:

```
SAS
```

When the request is received, the operating system executes a set of instructions that makes all the components of the SAS System (such as statements, procedures, formats, and functions) available to your SAS job. The operating system also provides space in the computer's main memory in which to execute the SAS program.

You can invoke SAS to execute in *batch mode*, *noninteractive mode*, or *interactive mode*. The mode determines how your program executes, as described below:

Batch Mode
: Submit a group of control language statements and SAS statements to the operating system. The control language and SAS statements may be stored in a file or may be in a stack of cards. When you submit the job, the operating system schedules (enqueues) the job for execution. Once execution begins, you cannot alter the job or the execution process.

Noninteractive Mode
: Create a file containing your SAS program only (with no control language). When you enter the command to invoke SAS from your terminal, you also enter the name of the file containing your SAS program. The operating system invokes SAS, locates the file, and executes the program. This mode is similar to batch in that once execution begins, you cannot alter the execution process. However, execution does proceed immediately; the job is not enqueued.

Interactive Mode
: (also called *conversational* mode) Issue a command to invoke SAS from your terminal. One of two things happens:

  - You will begin a *line-prompt session* in which SAS prompts you, usually with a line number and question mark, to enter your SAS program line by line. Your program executes one step at a time as you enter the statements.
  - You will begin a *display manager session* in which you enter the SAS program on the program editor screen and issue the SUBMIT command to submit the code to SAS.

  Whether your session is in line-prompt or display manager mode depends on the setting of the SAS system option DMS | NODMS and the kind of terminal you have. To use display manager, you must have a full-screen terminal and DMS must be in effect. Line-prompt mode occurs when your terminal is not full screen or when NODMS is in effect.

The **Details** section for your operating system illustrates how to invoke SAS. See **Modes of Execution** in the "SAS System Topics" chapter of the *SAS User's Guide: Basics* for more information on batch, noninteractive, and interactive execution modes. Also see the "SAS Display Manager" appendix of this book for information.

Receiving the Output

The set of instructions that the operating system executes when you invoke SAS includes requests telling the operating system where to send the output from the SAS job. This is called the *default destination*. SAS output consists of two main parts: 1) the SAS log, which contains the SAS statements used in the job, notes, and error messages; and 2) the SAS procedure output. All SAS jobs output a SAS log; however, not all SAS jobs produce procedure output.

The default destination of the SAS output depends on the execution mode:

- For a batch job the default destination is often a printer or a disk file.
- In noninteractive mode the SAS log and procedure output are displayed on the screen after the entire job executes.

- In interactive line-prompt mode, the SAS log and procedure output are displayed on the terminal screen as each step of your program executes instead of after the entire job executes.
- In interactive display manager mode, lines written to the SAS log appear on the SAS log screen; procedure output displays on the procedure output screen.

You can change the default destination for SAS output by using proper control language and SAS system options. (See **Re-routing SAS Output**, below). For more information on SAS output, refer to the chapter "Log and Procedure Output" in the *SAS User's Guide: Basics*.

Defining Files

In many SAS programs you will want to read or write a *permanent file* (a file that is stored for later use.) For example, you may want a SAS program to read a file of data that are stored on tape in order to create a SAS data set that can be printed by PROC PRINT, or you may want to enter new data after a CARDS statement and then store the data in a permanent disk file for later use.

A permanent file resides in a storage location managed by the operating system. Thus, the SAS System must interact with the operating system to access a permanent file. You allow this interaction to take place by *defining* the permanent file(s) needed by your program.

SAS programs use two general kinds of files: *SAS files* and *external files*. SAS files are specially structured files that can be created and processed only by the SAS System. SAS data sets are the most commonly used of the SAS files. External files can be created and processed by other programming languages, as well as by SAS. For the most part, SAS procedures use SAS files.

Following is a general description of how to define permanent SAS and external files. Refer to **Details** for your operating system for examples that illustrate how to define files for programs that read and create external files and SAS files.

External files To define a permanent external file, associate a fileref (short for "file reference") with the complete name of the external file. You then specify the fileref in SAS statements to refer to the external file. For example, if you are reading an external file, specify the fileref in the INFILE statement. If you are creating an external file, specify the fileref in the FILE statement.

In most cases you must explicitly associate the fileref with the complete name of the external file. The method used to associate the fileref with the file's name differs for each operating system, so be sure to see the **Details** section for your operating system.

SAS files To define a permanent SAS file, associate a *libref* (short for "library reference") with a permanent library of SAS files, called a *SAS data library*. The **Details** section for your operating system includes complete instructions on how to associate a libref with a SAS data library in your environment.

After the file is defined, you use the libref as the first-level name of the SAS file's complete two-level name in any SAS statement that requests the file. For example, the name could be specified in a PROC, DATA, or SET statement.

Once defined, a libref can be used repeatedly in SAS statements in that job or session to refer to any existing SAS file in the library or to add SAS files to the library.

In most cases librefs and filerefs must follow the rules for SAS names. Exceptions, if any, are given in **Details** for your operating system. For a general discussion of SAS files, their names, and SAS data libraries, see the chapter "SAS Files" in the *SAS User's Guide: Basics*.

SAS statements used to request files The following list contains a brief description of the SAS statements that are used to request files. If you use one of these statements to request a permanent file, the file must also be defined. See the next section for a list of SAS statements used to define permanent files. All of these statements are fully described in the *SAS User's Guide: Basics*:

DATA initiates the DATA step and allows you to create a SAS data set. If the data set is permanent, you must define the SAS library in which to store it unless your operating system is one that defines it for you (see **Details**).

FILE specifies a fileref that refers to an external file. Subsequent PUT statements write to this file. Refer to **Details** for your operating system for information on how to define an external file.

%INCLUDE specifies a fileref that refers to an external file containing SAS statements that you want to execute. Refer to **Details** for your operating system for information on how to define an external file.

INFILE specifies a fileref that refers to an external file to be read by the SAS program. Refer to **Details** for your operating system for information on how to define an external file.

PROC uses an option to refer to a SAS library or an external file that is to be read or created. For those procedures that can read and create files in the same execution, more than one option can be specified to refer to the appropriate SAS library or external file. When the procedure uses an entire SAS library, the value of the option is a libref. The value is a two-level permanent SAS file name when the procedure uses a SAS file within the library. If the procedure reads from or writes to an external file, the value of the option is a fileref.

SET
MERGE
UPDATE read and manipulate observations in SAS data sets. If a data set being accessed by one of these statements is permanent, you must define the SAS library in which it is stored unless your operating system is one that defines the library for you (see **Details**).

SAS statements used to define files The following list contains a brief description of the SAS statements that are used to define permanent files. Note that under some operating systems, only control language can be used to define a file, not SAS statements. Be sure to read the **Details** section for your operating system. All of these statements are fully described in the *SAS User's Guide: Basics*:

FILENAME specifies a fileref with a complete file name, which includes a directory name and the name of a specific file in the directory. If the FILENAME statement does not precede a SAS statement that specifies a fileref, the SAS System uses a file in the current default directory.

LIBNAME associates a libref with a directory name. This statement must precede a SAS statement that specifies the libref.

LIBSEARCH establishes a search list of directories by specifying a list of librefs. Each libref listed must be associated with a directory in a LIBNAME statement. LIBSEARCH must follow LIBNAME statements that define the directories.

X allows you to issue operating system commands from within your SAS program if executing in interactive or noninteractive mode. If the X statement is used to issue an operating system command to define a permanent file, it must precede the statement that refers to that file (see **Details**).

Re-routing SAS Output

As discussed above, the default destination of SAS output is determined by the mode of execution. You can change the default destination with the proper control language or with SAS options. For example, if you are executing SAS in display manager mode, by default the SAS log appears on the log screen, and the procedure output appears on the output screen. You can copy the SAS log and procedure output with the PRINT command of display manager.

When you re-route output, you must override default output handling. This endeavor is both operating system- and site-specific. The method you use will also depend on what you want to do. For example, you can route your output to a particular printer or terminal, or you can even route it to a permanent file. Use the following sources of information to learn how to route output:

- *SAS User's Guide: Basics*. Look for SAS system options used to re-route output on your operating system.
- SAS Companion or Technical Report for your operating system. Look for options and control language used to re-route output.
- The SAS consultant at your site.

DETAILS FOR THE AOS/VS OPERATING SYSTEM

The SAS System running under the AOS/VS operating system is documented in "Changes and Enhancements in the Base SAS and SAS/GRAPH Products under AOS/VS," SAS Technical Report P-129.

Invoking the SAS System

Enter the SAS command after you receive the) prompt:

```
) SAS
```

or

```
) SAS/options
```

or

```
) SAS filename
```

If you enter *SAS* you will invoke the SAS System in interactive mode. Use the second method to specify valid *options* when you invoke SAS. For example, you can specify *SAS/FSDEVICE=devicename* to invoke SAS in display manager mode. With the third method, you can specify *SAS filename* to give the name of a file containing the SAS program to be executed. This method is called *noninteractive mode*.

Note that the SAS command is a CLI macro file that contains a set of control language instructions to invoke the SAS System.

Read more about the SAS command in SAS Technical Report P-129. For information on executing SAS in batch mode see P-129 and the *AOS/VS CLI User's Manual*.

Defining Files

How you define a file depends on whether the file is a SAS or non-SAS (external) file. The method that you use allows the SAS System to distinguish between SAS files and external files.

External files You can define an external file in one of the following two ways:

- with the CREATE/LINK command of the CLI control language before you invoke the SAS System
- in the FILENAME statement within your SAS program.

With either method you associate a *fileref* with a complete file name. The complete file name includes the directory name and the name of a specific file within the directory. You can also issue the CREATE/LINK command in the X statement, a SAS statement that allows you to enter CLI commands in your SAS program.

The fileref is then used in SAS statements, such as FILE and INFILE, to refer to the external file. If you specify a fileref in a SAS statement before you define the external file, the SAS System reads or creates a file in the current default directory with the following name:

```
:UDD:currentdefaultdirectory:fileref
```

where *fileref* is specified in the INFILE statement to give the name of the file that is read from the current default directory or where *fileref* is specified in the FILE statement to give the name of the file being created in the current default directory.

SAS files If you want to read or create a SAS file, you must define the appropriate directory with the LIBNAME statement. In the LIBNAME statement you associate a *libref* with a directory name only. The libref is then specified in SAS statements, such as DATA, SET, MERGE, and so on, to refer to the directory. Once you have defined a directory in the LIBNAME statement, you can use the associated libref to refer to any file in that directory.

If you do not specify the LIBNAME statement before you use the libref in a SAS statement, you will receive an error message that the directory to which you are referring cannot be found.

Note: discussion of SAS files in the following examples is limited to the most commonly used type, a **SAS data set**. A SAS data set is a SAS file containing observations and variables.

Methods for defining SAS data sets and external files are illustrated in the following sections.

Reading an External File

The following program reads an external file and includes a FILENAME statement to define the external file:

```
) SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY ':UDD:YOURDIR1:YOUR.RAW.DATA';
2? DATA WEATHER;
3?    INFILE DAILY;
more SAS statements
```

```
7? ENDSAS;
```

The FILENAME statement associates the fileref DAILY with the complete file name, and DAILY is then specified in the INFILE statement. Note that the complete file name includes a directory named YOURDIR1 and the file YOUR.RAW.DATA within the directory.

You can also use the CREATE/LINK to define the external file before you invoke the SAS System. For example:

```
CREATE/LINK DAILY :UDD:YOURDIR1:YOUR.RAW.DATA
```

If you use the CREATE/LINK command, you do not need the FILENAME statement. If you chose to define the file by issuing the CREATE/LINK command with the X statement, you can simply replace the FILENAME statement in the above program with the following SAS statement:

```
X 'CREATE/LINK DAILY :UDD:YOURDIR1:YOUR.RAW.DATA';
```

Since the X statement is a SAS statement, be sure to put a semicolon at the end.

If you do not define the external file, the SAS System will read (or attempt to read) a file from the current default directory with the following name:

```
:UDD:currentdefaultdirectory:DAILY
```

Reading a SAS Data Set

Use the LIBNAME statement to define the directory containing the SAS data set to be read, as shown in the following program:

```
) SAS (invoke the SAS System)
SAS messages
1? LIBNAME REPORT ':UDD:DIR2';
2? PROC PRINT DATA=REPORT.STATION;
3?    VAR A B C;
more SAS messages
4? ENDSAS;
```

The LIBNAME statement associates the libref REPORT with the directory :UDD:DIR2. REPORT is then used as the first level of the permanent SAS data set name REPORT.STATION in the PROC PRINT statement. When this program executes, the SAS System reads a SAS data set named STATION from the directory named :UDD:DIR2 and prints the values for variables A, B, and C.

If the LIBNAME statement is omitted or does not precede the PROC PRINT statement, you will receive an error message stating that the directory cannot be found.

Creating a SAS Data Set

Use the LIBNAME statement to define a directory in which to store a newly created SAS data set. The following program creates a SAS data set using an external file as input:

```
) SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY ':UDD:YOURDIR1:YOUR.RAW.DATA';
2? LIBNAME MAP ':UDD:YOURDIR2';
3? DATA MAP.WEATHER;
4?    INFILE DAILY;
more SAS statements and messages
```

```
5? ENDSAS;
```

The LIBNAME statement associates the libref MAP with YOURDIR2, the directory in which the SAS data set WEATHER is stored. MAP is then used as the first level of the permanent SAS data set name in the DATA statement. The FILENAME statement associates DAILY with the directory YOURDIR1 and the file YOUR. RAW.DATA, the input file.

If the LIBNAME statement is omitted, you will receive an error message stating that the directory cannot be found. If you do not specify the FILENAME statement, the SAS System will search the current default directory for a file named DAILY.

Creating an External File

Since the following program creates an external file, use the FILENAME statement to associate the fileref with the complete file name. The complete file name includes the directory name and the name of the file that is created within that directory. Since the following program uses a SAS data set as input, you must use a LIBNAME statement to define the appropriate directory:

```
) SAS (invoke the SAS System)
(SAS messages)
1? LIBNAME MAP ':UDD:DIR2'
2? FILENAME FREEZE ':UDD:DIR3:COLD.RAW.DATA';
3? DATA _NULL_;
4?    SET MAP.WEATHER;
5?    FILE FREEZE;
6?    IF TEMP < 32 THEN PUT TEMP;
more SAS statements and messages
9? ENDSAS;
```

The FILENAME statement associates FREEZE with the complete file name :UDD:DIR3:COLD.RAW.DATA. FREEZE is then specified in the FILE statement. You can also issue the CREATE/LINK command (before you invoke the SAS System or in the X statement) to define the external file.

If you do not define the external file for the above program, the SAS System will create a file in the current default directory with the following name:

```
:UDD:currentdefaultdirectory:FREEZE
```

DETAILS FOR THE CMS OPERATING SYSTEM

The SAS System running under the CMS operating System is documented in the *SAS Companion for the VM/CMS Operating System*.

Invoking the SAS System

Enter the SAS command after you receive the READY or R; prompt:

```
R;   (or READY)
SAS filename (options)
```

If your SAS program is stored in a file, specify the *filename* after the SAS command. Invoking the SAS System to execute in this way is called *noninteractive mode*. If you want to invoke SAS in interactive mode do not specify a filename. If you want to specify any of the valid *options* for the SAS command, enter a right parenthesis, followed by the options. The DMS | NODMS option and the type of termi-

nal you are using determine whether you enter a line-prompt or display manager session. If you have a full-screen terminal and DMS is in effect, you enter display manager. If your terminal is not full screen or NODMS is in effect, you enter a line-prompt interactive session.

Note that the SAS command is a CMS command created by the SAS Institute that contains a set of control language instructions to invoke the SAS System.

Read more about the SAS command in the *SAS Companion for the VM/CMS Operating System*. To learn about running a SAS job in a CMS batch machine, refer to the *IBM Virtual/System Product: CMS User's Guide*.

Defining Files

The CMS command FILEDEF is used to define files for use in SAS programs. The FILEDEF command associates a *libref* with a SAS file or a *fileref* with an external (non-SAS) file. The libref or fileref is called a *DDname* in CMS terminology. The form of the FILEDEF command is

```
FILEDEF DDname device filename filetype filemode(options)
```

where

- *DDname* serves as a SAS file's libref or an external file's fileref
- *device* indicates the file's storage medium (usually DISK or TAPE)
- *filename* and *filetype* are the names by which CMS knows the file
- *filemode* is a letter (or letter/number combination) indicating the minidisk on which the file is stored (for disk files only)
- *options* are any of a number of CMS options for the FILEDEF command.

In most cases, you do not have to issue the FILEDEF explicitly to define a SAS file because the SAS System issues the FILEDEF automatically. You must explicitly issue the FILEDEF for an external file.

A FILEDEF can be issued before the SAS System is invoked or after SAS is invoked in an X or CMS statement. In any case, the FILEDEF must be issued before the SAS statement that references the file.

Special Naming Conventions

Although the SAS System allows you to use an underscore (_) in SAS names, CMS does not. Therefore, do not use an underscore character in a libref or a fileref.

Reading an External File

Issue a FILEDEF command to associate a fileref with the external file to be read. Then, use the fileref in the INFILE statement of your SAS program. For example, suppose you write a DATA step that reads the external disk file CLIMATE DATA, which is stored on your A-disk, and you want to use DAILY as the fileref:

```
R;
FILEDEF DAILY DISK CLIMATE DATA A
R;
SAS (invoke the SAS System)
(SAS messages)
1?
DATA WEATHER;
2?
INFILE DAILY;
3?
INPUT TEMP PRECIP SUNHRS;
```

```
4?
more SAS statements
```

The fileref DAILY points to the CMS file CLIMATE DATA A.

Reading a SAS Data Set

Under most circumstances you do not need to define a libref explicitly for a SAS file because SAS makes the association automatically. To access an existing SAS file on any minidisk, just use the file's two-level name in the appropriate SAS statement. SAS searches all accessed minidisks for the file you have specified and issues the appropriate FILEDEF when it finds the file. For example, suppose the SAS data set FOOD.PRICES is stored on your B-disk. When SAS reads this statement:

```
SET FOOD.PRICES;
```

it searches all accessed minidisks for a data set with filename PRICES and filetype FOOD. When it finds PRICES FOOD B, this FILEDEF is automatically issued:

```
FILEDEF FOOD DISK PRICES FOOD B
```

Note: if you have multiple SAS files with the same filename and filetype on different minidisks, SAS will find only the file on the minidisk that is first in the search order. For example, if you have a PRICES FOOD A and a PRICES FOOD B, SAS will find PRICES FOOD A. You must issue a FILEDEF for PRICES FOOD B explicitly if you want to access it.

Creating a SAS Data Set

Under most circumstances, you do not need to define a libref explicitly to create a SAS file because SAS makes the association automatically. When you are creating a new SAS file and you specify a two-level SAS name, SAS automatically issues a FILEDEF using the libref (first-level name) for the DDname parameter **and** for the filetype unless a FILEDEF with that DDname is already in effect. By default, the data set will be written to your A-disk. For example, suppose you are creating a SAS data set called FOOD.PRICES:

```
DATA FOOD.PRICES;
  INPUT ... ;
more SAS statements
```

When SAS reads the name FOOD.PRICES, it checks to see if a FILEDEF with DDname FOOD has been issued. If there is no such FILEDEF, SAS issues one automatically, using FOOD for both the DDname and filetype, PRICES as the filename, and A as the filemode:

```
FILEDEF FOOD DISK PRICES FOOD A
```

Note: do not use the same filetype for SAS files and external (non-SAS) files.

If you want a new SAS data set to be stored on a disk other than your A-disk, you must issue the appropriate FILEDEF explicitly. Specify the appropriate values for the DDname and filemode parameters. You can specify any value for filename and filetype; SAS will substitute the libref (first-level name) and data set name (second-level name) for filetype and filename, respectively. For example, if you want to store FOOD.PRICES on a B-disk, your FILEDEF could be:

```
FILEDEF FOOD DISK DUMMY DUMMY B
```

Notice that CMS and SAS identify the file by the same two names, but the order is reversed. For example, SAS calls the data set FOOD.PRICES, and CMS calls it PRICES FOOD.

Creating an External File

To create an external file in a SAS program, first issue a FILEDEF command to associate a fileref with the name of the external file to be created. Then use the fileref in the FILE statement of your SAS program. For example, suppose you write a DATA step that reads the external disk file CLIMATE DATA, (which is stored on your A-disk) and creates a second external file called COLD TEMPS A. You use the filerefs DAILY and FREEZE to reference the files:

```
R;
FILEDEF DAILY DISK CLIMATE DATA A
R;
FILEDEF FREEZE DISK COLD TEMPS A
R;
SAS
1?
DATA _NULL_;
2?
INFILE DAILY;
3?
INPUT TEMP PRECIP SUNHRS;
4?
FILE FREEZE;
5?
more SAS statements
```

DETAILS FOR THE OS OPERATING SYSTEM AND TSO

The SAS System running under the OS operating system is documented in the *SAS Companion for the OS Operating System and TSO*.

OS Batch

Invoking the SAS System Use the EXEC statement in your JCL:

```
// EXEC SAS,OPTIONS='options'
```

In this case, the OS cataloged procedure named *SAS* contains all the job control language instructions to invoke the SAS System in batch mode. The *OPTIONS=* parameter can be used to specify valid SAS system options.

Read more about the SAS cataloged procedure in the *SAS Companion for OS Operating System and TSO*.

Defining files Use the DD (Data Definition) statement to define permanent files, both SAS and external files. The DD statement associates a *fileref* with a permanent external file or a *libref* with a permanent library of SAS files. In OS terminology the fileref or the libref is called a *DDname*.

The external file can be an OS sequential data set or a partitioned data set (PDS). A sequential data set contains records stored in a sequence under a unique name. If the external file is a sequential data set, you associate a fileref with the sequential data set name in the DD statement, as follows:

```
//fileref DD DSN=external.file.name,DISP=disposition
```

The *fileref* is then specified in SAS statements, such as INFILE and FILE, to refer to the external file. For example,

```
INFILE fileref;
```

A PDS contains a group of data sets called members. The PDS has a name that identifies the group. Each member within the PDS has a unique name. You can identify one member by enclosing the member name in parentheses and including it as the last level of the PDS name. For example, if you want to define an external file that is a member of a PDS, you must specify the DD statement as follows:

```
//fileref DD DSN=external.file.name(member),DISP=disposition
```

In this case you only have access to the member specified. Then, you use the fileref in SAS statements, such as INFILE and FILE, to refer to that member.

To have access to all members of the PDS, use the PDS name only (without a member name) in the DD statement, as follows:

```
//fileref DD DSN=external.file.name,DISP=disposition
```

When you define the PDS in this way, you include the member name in parentheses after the fileref to refer to the specific member in the PDS. For example,

```
INFILE fileref(member1);
FILE fileref(member2);
```

The above statements refer to two members of the same PDS.

If you use a fileref in a SAS statement that is not associated with an external file in the DD statement, you will receive an error message indicating that the file to which you are referring has not been defined.

A group of SAS files is called a *SAS data library*. Once defined, all SAS files within the library are accessible. For example,

```
//libref DD DSN=SAS.library.name,DISP=disposition
```

Then use the *libref* in SAS statements, such as DATA, SET, and MERGE, to refer to the SAS library. You refer to a specific file within the library by including the SAS file name after the libref. The libref and the SAS file name are always separated by a period. For example,

```
DATA libref.SASname;
```

In some SAS statements, such as certain PROC statements, you use only the libref to refer to the entire SAS library. The utility procedure DATASETS is a good example:

```
PROC DATASETS LIBRARY=libref;
```

All of the SAS files in the library referred to by the libref are available to be processed by the DATASETS procedure.

A SAS library can contain several different types of SAS files. The discussion of SAS files in this section is limited to the most commonly used type, a **SAS data set**. A SAS data set is a SAS file that contains observations and variables.

If you use a libref in a SAS statement that is not associated with a SAS library in the DD statement, you will receive an error message indicating that the file to which you are referring has not been defined.

The DISP= parameter Each of the DD statement examples includes a DISP= parameter. DISP= specifies a disposition (status) for the file at the beginning and end of the job. The disposition also implies an access method. (Refer to the *SAS Companion for OS Operating Systems and TSO* for details on specifying the DISP= parameter in the DD statement.) All of the examples in the following sections use existing files and, therefore, specify either DISP=OLD to obtain exclusive access to an existing file or DISP=SHR to obtain shared access to an existing file.

If your SAS program writes to an existing SAS library, you must specify DISP=OLD in the DD statement because you are updating the library. You must have exclusive access to a SAS library to update it. The need to specify DISP=OLD is noted for the following examples, where appropriate.

Methods for defining SAS libraries and external files are illustrated in the following sections.

Reading an external file To read an external file you must include a DD statement in the JCL used to invoke the SAS System. In the following job:

```
//jobname JOB accountinginformation
// EXEC SAS
//DAILY DD DSN=YOUR.RAW.DATA(MON),DISP=SHR
//SYSIN  DD  *
DATA WEATHER;
   INFILE DAILY;
   INPUT TEMP PRECIP SUNHRS;
more SAS statements
```

the DD statement associates the fileref DAILY with the external file YOUR.RAW.DATA(MON). Notice that the external file is a PDS and the member MON is specified. DAILY is then used in the INFILE statement to refer to the external file.

Reading a SAS data set You also use the DD statement to define a permanent SAS library. The DD statement in the following job associates the libref REPORT with the SAS library named YOUR.SAS.LIB: REPORT is then specified as the first level of a permanent SAS data set name in the PROC PRINT statement.

```
//jobname JOB accountinginformation
// EXEC SAS
//REPORT DD DSN=YOUR.SAS.LIB,DISP=SHR
//SYSIN DD *
PROC PRINT DATA=REPORT.STATION;
   VAR A B C;
```

The association is necessary to allow the SAS System to read a SAS data set named STATION from the SAS library YOUR.SAS.LIB. When the program executes, PROC PRINT locates STATION and prints the values of the variables A, B, and C.

Note: in the DD statement YOUR.SAS.LIB is an existing SAS library defined with DISP=SHR to allow for shared use of the SAS library that is being read.

Creating a SAS data set The following program creates a SAS data set using an external file as input. Use the DD statement to define the existing SAS library in which to store the SAS data set created by this program. You must also define the external file in a DD statement:

```
//jobname JOB accountinginformation
// EXEC SAS
//DAILY DD DSN=YOUR.RAW.DATA(MON),DISP=SHR
//MAP DD DSN=YOUR.SAS.LIB,DISP=OLD
//SYSIN DD *
DATA MAP.WEATHER;
   INFILE DAILY;
more SAS statements
```

The first DD statement associates the fileref DAILY with the permanent external file named YOUR.RAW.DATA(MON). Notice that you again refer to a specific

member of a PDS. The second DD statement associates the libref MAP with the existing SAS library named YOUR.SAS.LIB. MAP is then used as the first level of the permanent SAS data set name in the DATA statement. The SAS data set WEATHER created by this program is stored in YOUR.SAS.LIB.

Note: in contrast to the previous example, the SAS library must be defined with DISP=OLD because it is existing and because the SAS System will not allow you to write to a SAS library defined with shared access (DISP=SHR). Remember, if your program updates a SAS library, you must have exclusive access to the library.

Creating an external file If your SAS program creates an external file, you must use a DD statement to define the file. The following program creates an external file using a SAS data set as input; therefore, you must include two DD statements in the JCL used to invoke the SAS System:

```
//jobname JOB accountinginformation
// EXEC SAS
//MAP DD DSN=YOUR.SAS.LIB,DISP=SHR
//FREEZE DD DSN=COLD.RAW.DATA,DISP=OLD
//SYSIN  DD  *
DATA _NULL_;
   SET MAP.WEATHER;
   FILE FREEZE(JAN);
   IF TEMP < 32 THEN PUT TEMP;
more SAS statements
```

In this case the libref MAP refers to a file named YOUR.SAS.LIB, and MAP is then used as the first level of the permanent SAS data set name in the SET statement. The fileref FREEZE refers to a PDS named COLD.RAW.DATA. The member of the PDS (JAN) created by this program is specified in the FILE statement. FREEZE is then used in the FILE statement. When this program executes, the PUT statement writes all TEMP values less than 32 to the external file COLD.RAW.DATA(JAN).

TSO (OS Interactive)

Invoking the SAS System Enter the TSO command after you receive the READY prompt:

```
READY
SAS OPTIONS('options')
```

or

```
SAS INPUT('''filename''')
```

Enter *SAS OPTIONS('options')* to invoke SAS in interactive mode and specify valid SAS *options*. The DMS | NODMS option and the type of terminal you use determine whether you enter a line-prompt or display manager session. If you use a full-screen terminal and if DMS is in effect, you enter a display manager session. If your terminal is not full screen or if NODMS is in effect, you enter a line-prompt session.

Enter *SAS INPUT('''filename''')* to invoke SAS in noninteractive mode and specify *filename* to give the name of the file containing the SAS program to be executed.

The SAS command is a TSO CLIST (Command LIST) that contains a set of control language instructions to invoke the SAS System.

Defining files Use the ALLOCATE command of TSO to define permanent files—SAS files and external files. The ALLOCATE command associates a *fileref* with a

permanent external file or a *libref* with a permanent library of SAS files. In TSO terminology a libref or fileref is called a *DDname*.

The external file can be an OS sequential data set or a partitioned data set (PDS). A sequential data set contains records stored in a sequence under a unique name. If the external file to be defined is a sequential data set, you associate a fileref with the sequential data set name in the ALLOCATE command, as follows:

```
ALLOCATE FILE(fileref) DATASET('external.file.name')disposition
```

The *fileref* is then specified in SAS statements, such as INFILE and FILE, to refer to the external file. For example,

```
INFILE fileref;
```

A PDS contains a group of data sets called members. The PDS has a name that identifies the group. Each member within the PDS has a unique name. You can identify one member by enclosing the member name in parentheses and including it as the last level of the PDS name. For example, if you want to define an external file that is a member of a PDS, you must specify the ALLOCATE command as follows:

```
ALLOCATE FILE(fileref)DATASET('external.file.name.(member)') disposition
```

In this case you only have access to the member specified. Then you use the fileref in SAS statements, such as INFILE and FILE, to refer to that member.

To have access to all members, give the PDS name only (without a member name) in the ALLOCATE command, as follows:

```
ALLOCATE FILE(fileref) DATASET('external.file.name') disposition
```

When you use the fileref in SAS statements, such as INFILE and FILE, you include the member name in parentheses after the fileref to refer to a specific member in the PDS. For example,

```
INFILE fileref(member1);
FILE fileref(member2);
```

The above statements refer to two members of the same PDS.

If you use a fileref in a SAS statement that is not associated with an external file in the ALLOCATE command, you will receive an error message indicating that the file to which you are referring has not been defined.

A group of SAS files is called a *SAS data library*. Once defined, all SAS files within the library can be accessed. For example,

```
ALLOCATE FILE(libref) DATASET('SAS.library.name')disposition
```

Then use the *libref* in SAS statements, such as DATA, SET, and MERGE, to refer to the SAS library. You refer to a specific file within the library by including the SAS file name after the libref. The libref and the SAS file name are always separated by a period. For example,

```
DATA libref.SASname;
```

In some SAS statements, such as certain PROC statements, you use only the libref to refer to the entire SAS library. The utility procedure DATASETS is a good example:

```
PROC DATASETS LIBRARY=libref;
```

All of the SAS files in the library referred to by the libref are available to be processed by the DATASETS procedure.

A SAS library can contain several different types of SAS files. The discussion of SAS files in this section is limited to one type, **SAS data sets**. A SAS data set is a SAS file that contains observations and variables.

If you use a libref in a SAS statement that is not associated with a SAS library in the ALLOCATE command, you will receive an error message indicating that the file to which you are referring has not been defined.

The disposition operand Each ALLOCATE command example includes a disposition operand. *Disposition* specifies the status of the file at the beginning and end of the job and also implies an access method. (Refer to the *SAS Companion for OS Operating Systems and TSO* for details on specifying the disposition operand.) All of the examples in this section use existing files and, therefore, specify either OLD to obtain exclusive access to an existing file or SHR to obtain shared access to an existing file.

If your SAS program writes to an existing SAS library, you must specify a disposition of OLD in the ALLOCATE command because you are updating the library. You must have exclusive access to a SAS library to update it. The need to specify OLD is noted for the following examples, where appropriate.

If your TSO CLIST invokes the SASCP command, you can also issue the ALLOCATE command from within your SAS program with the X or TSO statement. If you use the X or TSO statement, be sure to include it before the SAS statement that requests the file. For example, the X statement that issues an ALLOCATE command to define an input file must be specified before the INFILE statement.

If you do not define the permanent files requested by your SAS program, you will receive an error message.

Methods for defining SAS libraries and external files are illustrated in the following sections.

Reading an external file You can enter the ALLOCATE command before you invoke the SAS System to define the file to be read by the following program:

```
ALLOCATE FILE(DAILY) DATASET('YOUR.RAW.DATA(MON)') SHR
SAS  (invoke the SAS System)
SAS messages
1? DATA WEATHER;
2?   INFILE DAILY;
more SAS statements
6? ENDSAS;
```

The ALLOCATE command associates the fileref DAILY with the permanent external file YOUR.RAW.DATA(MON). Notice that you are defining a specific member of a PDS. DAILY is then specified in the INFILE statement. You can also define the file by issuing the ALLOCATE command in the X statement, as follows:

```
X ALLOCATE FILE(DAILY) DATASET('YOUR.RAW.DATA(MON)') SHR;
```

The X statement must precede the INFILE statement in the above program; otherwise, you will receive an error message indicating that you are referring to a file that has not been defined.

Reading a SAS data set Use the ALLOCATE statement to define a permanent SAS library to be used as input. In the following example the ALLOCATE command associates the libref REPORT with the SAS library named YOUR.SAS.LIB. REPORT is then used as the first level of the permanent SAS data set name in the PROC PRINT statement:

```
SAS  (invoke the SAS System)
SAS messages
```

```
1? TSO ALLOCATE FILE(REPORT) DATASET('YOUR.SAS.LIB') SHR;
2? PROC PRINT DATA=REPORT.STATION;
3?    VAR A B C;
SAS messages
4? ENDSAS;
```

In this case, the TSO statement (the equivalent of the X statement under TSO) is used to issue the ALLOCATE command. When this program executes, PROC PRINT locates the data set named STATION and prints the values of the variables A, B, and C.

Creating a SAS data set The following program creates a SAS data set using a permanent external file as input. Therefore, you must issue the ALLOCATE command twice: once to define the external file to be used as input, and once to define the SAS library in which to store WEATHER, the SAS data set created by this program:

```
ALLOCATE FILE(DAILY) DATASET('YOUR.RAW.DATA(MON)') SHR
ALLOCATE FILE(MAP) DATASET('YOUR.SAS.LIB') OLD
SAS  (invoke the SAS System)
SAS messages
1? DATA MAP.WEATHER;
2?   INFILE DAILY;
3?   INPUT TEMP PRECIP SUNHRS;
more SAS statements and messages
6? ENDSAS;
```

The first ALLOCATE command associates the fileref DAILY with the external file YOUR.RAW.DATA(MON), a specific PDS member. DAILY is then used in the INFILE statement. The second ALLOCATE command associates the libref MAP with the SAS library YOUR.SAS.LIB. MAP is then used as the first level of the permanent SAS data set name in the DATA statement. When this program executes, the permanent SAS data set WEATHER contains observations with variables TEMP, PRECIP, and SUNHRS.

Note: this program creates a new SAS data set in the SAS library YOUR.SAS.LIB and thereby updates the library. Notice that the YOUR.SAS.LIB is defined in the ALLOCATE command with a disposition of OLD.

Creating an external file The following program creates an external file using a SAS data set as input. As before, you must issue the ALLOCATE command twice: once to define the permanent external file created by the program and once to define the permanent SAS library that is used as input:

```
ALLOCATE FILE(MAP) DATASET('YOUR.SAS.LIB') SHR
ALLOCATE FILE(FREEZE) DATASET('COLD.RAW.DATA') OLD
SAS  (invoke the SAS System)
SAS messages
1? DATA _NULL_;
2?   SET MAP.WEATHER;
3?   FILE FREEZE(JAN);
4?   IF TEMP < 32 THEN PUT TEMP;
more SAS statements
8? ENDSAS;
```

The first ALLOCATE command associates the libref MAP with the SAS library named YOUR.SAS.LIB. MAP is then used in the SET statement as the first level in the permanent SAS data set name MAP.WEATHER. The second ALLOCATE command associates the fileref FREEZE with a PDS named COLD.RAW.DATA.

FREEZE is then used in the FILE statement along with the PDS member JAN that is created by this program. When the program executes, the SAS data set WEATHER is read from the SAS library YOUR.SAS.LIB and all TEMP values less than 32 are written to the external file COLD.RAW.DATA(JAN).

DETAILS FOR THE PRIMOS OPERATING SYSTEM

The SAS System running under the PRIMOS operating system is documented in "Changes and Enhancements in the Base SAS and SAS/GRAPH Products under PRIMOS," SAS Technical Report P-130.

Invoking the SAS System

Enter the SAS command after you receive the OK, prompt:

```
OK, SAS
```

or

```
OK, SAS -options
```

or

```
OK, SAS filename
```

If you enter *SAS* you will invoke the SAS System in interactive mode. Use the second method to specify valid *options* when you invoke SAS. For example, you can specify *SAS -FSDEVICE=devicename* to invoke SAS in display manager mode. With the third method, you can specify *filename* to give the name of a file containing the SAS program to be executed. This method is called *noninteractive mode*.

Read more about the SAS command in SAS Technical Report P-130.

For information on executing SAS in batch mode, refer to P-130 and *PRIMOS Command Reference Guide*.

Defining Files

How you define a file depends on whether the file is a SAS or non-SAS (external) file. The method that you use allows the SAS System to distinguish between SAS files and external files.

External files Use the FILENAME statement to define an external file. In the FILENAME statement you associate the *fileref* with a complete file name, which includes the directory name and a specific file within the directory. The FILENAME statement must be included in the SAS program before a SAS statement that refers to the permanent external file. For example, if your program uses an external file as input, the FILENAME statement that defines the file must precede the INFILE statement.

If you do not define the external file to be read or created by your program, the SAS System reads or creates a file in the current directory with the following name:

```
<masterfiledirectory>currentdirectory>fileref
```

where *fileref* is specified in the INFILE statement to give the name of the file that is read from the current default directory or where *fileref* is specified in the FILE statement to give the name of the file being created in the current default directory.

SAS files If you want to read or create a permanent SAS file, you must define the appropriate directory with the LIBNAME statement. In the LIBNAME state-

ment you associate a *libref* with a directory name only. The libref is then used in SAS statements, such as DATA, SET, MERGE, and so on, to refer to the directory. Once you have defined a directory in the LIBNAME statement, you can use the associated libref to refer to any file in that directory.

The LIBNAME statement must precede a SAS statement that refers to the SAS files (that is, a statement that specifies the libref); otherwise, you will get an error message stating that the directory to which you are referring cannot be found.

Note: when you specify the directory name in the LIBNAME statement or the complete file name (directory name and specific file in the directory) in the FILENAME statement, consider including the master file directory name in the specification. For example,

```
LIBNAME libref '<MASTERDIR>YOURDIR';
FILENAME fileref '<MASTERDIR>YOURDIR2>INPUT.STUFF';
```

Although PRIMOS does not require this level of specification, including the name of the master file directory in a directory specification will ensure that you get the file you want.

Note: discussion of SAS files in the following examples is limited to the most commonly used type, a **SAS data set**. A SAS data set is a SAS file containing observations and variables.

Methods for defining SAS data sets and external files are illustrated in the following sections.

Reading an External File

Use the FILENAME statement in the SAS program to define the file to be read. The FILENAME statement associates the fileref DAILY with the complete file name <MFD>YOURDIR1>YOUR.RAW.DATA:

```
OK, SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY '<MFD>YOURDIR1>YOUR.RAW.DATA';
2? DATA WEATHER;
3?    INFILE DAILY;
more SAS statements
7? ENDSAS;
```

DAILY is then used in the INFILE statement. If you do not specify the FILENAME statement, the SAS System searches the current directory for the following file:

```
<masterfiledirectory>currentdirectory>DAILY
```

Reading a SAS data set

Use the LIBNAME statement to define the directory containing the SAS data set to be read:

```
OK, SAS (invoke the SAS System)
SAS messages
1? LIBNAME REPORT '<MFD>DIR2';
2? PROC PRINT DATA=REPORT.STATION;
3?    VAR A B C;
more SAS messages
4? ENDSAS;
```

In the above program the LIBNAME statement associates the libref REPORT with the directory named <MFD>DIR2. REPORT is then used as the first level of the permanent SAS data set name REPORT.STATION in the PROC PRINT state-

ment. This association is necessary to allow PROC PRINT to locate a SAS data set named STATION in this directory and print the values of variables A, B, and C.

If the LIBNAME statement is omitted or is not included before the SAS statement that specifies the libref, the SAS System will issue an error message that the directory cannot be found.

Creating a SAS Data Set

The following program creates a permanent SAS data set named WEATHER. You must use the LIBNAME statement to define the directory in which to store WEATHER. This program also uses a external file as input:

```
OK, SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY '<MFD>YOURDIR1>YOUR.RAW.DATA';
2? LIBNAME MAP '<MFD>YOURDIR2';
3? DATA MAP.WEATHER;
4?    INFILE DAILY;
more SAS statements and messages
5? ENDSAS;
```

In this case LIBNAME associates the libref MAP with the directory <MFD>YOURDIR2. MAP is then used as the first level of the SAS data set name in the DATA statement.

If the LIBNAME statement is omitted, you will receive an error message stating that the directory cannot be found. If you do not specify the FILENAME statement, the SAS System will search the current directory for a file named DAILY.

Creating an External File

The following program creates an external file using a SAS data set as input. Use the FILENAME statement to associate the fileref with the complete file name. The complete file name will include the directory name and the name of a file to be created in that directory. Use a LIBNAME statement to define the directory containing the SAS data set to be read:

```
OK, SAS (invoke the SAS System)
SAS messages
1? LIBNAME MAP '<MFD>YOURDIR2'
2? FILENAME FREEZE '<MFD>DIR3>COLD.RAW.DATA';
3? DATA _NULL_;
4?    SET MAP.WEATHER;
5?    FILE FREEZE;
6?    IF TEMP < 32 THEN PUT TEMP;
more SAS statements and messages
9? ENDSAS;
```

The LIBNAME statement associates the libref MAP with <MFD>YOURDIR2, the directory containing the SAS data set WEATHER. MAP is then used as the first level of a permanent SAS data set name in the SET statement.

When this program executes, all TEMP values less than 32 in the SAS data set WEATHER are written to an external file named COLD.RAW.DATA. This file is located in the directory <MFD>DIR3. If the FILENAME statement is omitted, the SAS System writes the values to a file in the current directory with the following name:

```
<masterfiledirectory>currentdirectory>FREEZE
```

DETAILS FOR THE VM/PC OPERATING SYSTEM

The SAS System running under the VM/PC operating system is documented in the *SAS Companion for the VM/CMS Operating System.*

Invoking the SAS System

Enter the SAS command after you receive the READY or R; prompt:

```
R;   (or READY)
SAS filename (options)
```

If your SAS program is stored in a file, specify the *filename* after the SAS command. Invoking the SAS System to execute in this way is called *noninteractive mode*. If you want to invoke SAS in interactive mode do not specify a filename. If you want to specify any of the valid *options* for the SAS command, enter a right parenthesis, followed by the options. The DMS | NODMS option and the type of terminal you are using determine whether you enter a line-prompt or display manager session. If you are using a full-screen terminal and DMS is in effect, you enter a display manager session. If your terminal is not full screen or if NODMS is in effect, you enter a line-prompt session.

Note that the SAS command is a CMS command created by SAS Institute that contains a set of control language instructions to invoke the SAS System.

Read more about the SAS command in the *SAS Companion for the VM/CMS Operating System.*

Defining Files

The CMS command FILEDEF is used to define files for use in SAS programs. The FILEDEF command associates a *libref* with a SAS file or a *fileref* with an external (non-SAS) file. The libref or fileref is called a *DDname* in CMS terminology. The form of the FILEDEF command is

```
FILEDEF DDname device filename filetype filemode(options
```

where

- *DDname* serves as a SAS file's libref or an external file's fileref
- *device* indicates the file's storage medium (usually DISK)
- *filename* and *filetype* are the names by which CMS knows the file
- *filemode* is a letter (or letter/number combination) indicating the minidisk on which the file is stored (for disk files only)
- *options* are any of a number of CMS options for the FILEDEF command.

In most cases, you do not have to issue the FILEDEF explicitly to define a SAS file because the SAS System issues the FILEDEF automatically. You do have to issue the FILEDEF explicitly for an external file.

A FILEDEF can be issued before the SAS System is invoked or after SAS is invoked in an X or CMS statement. In any case, the FILEDEF must be issued before the SAS statement that references the file.

Special Naming Conventions

Although the SAS System allows you to use an underscore (_) in SAS names, VM/PC does not. Therefore, do no use an underscore character in a libref or a fileref.

Reading an External File

Issue a FILEDEF command to associate a fileref with the external file to be read. Then use the fileref in the INFILE statement of your SAS program. For example, suppose you write a DATA step that reads the external disk file CLIMATE DATA, which is stored on your A-disk, and you want to use DAILY as the fileref:

```
R;
FILEDEF DAILY DISK CLIMATE DATA A
R;
SAS (invoke the SAS System)
(SAS messages)
1?
DATA WEATHER;
2?
INFILE DAILY;
3?
INPUT TEMP PRECIP SUNHRS;
4?
more SAS statements
```

The fileref DAILY points to the CMS file CLIMATE DATA A.

Reading a SAS Data Set

Under most circumstances, you do not need to define a libref explicitly for a SAS file because SAS makes the association automatically. To access an existing SAS file on any minidisk, just use the file's two-level name in the appropriate SAS statement. SAS searches all accessed minidisks for the file you have specified and issues the appropriate FILEDEF when it finds the file. For example, suppose the SAS data set FOOD.PRICES is stored on your B-disk. When SAS reads this statement,

```
SET FOOD.PRICES;
```

it searches all accessed minidisks for a data set with filename PRICES and filetype FOOD. When it finds PRICES FOOD B, this FILEDEF is automatically issued:

```
FILEDEF FOOD DISK PRICES FOOD B
```

Note: if you have multiple SAS files with the same filename and filetype on different minidisks, SAS will find only the file on the minidisk that is first in the search order. For example, if you have a PRICES FOOD A and a PRICES FOOD B, SAS will find PRICES FOOD A. You must issue a FILEDEF for PRICES FOOD B explicitly if you want to access it.

Creating a SAS Data Set

Under most circumstances, you do not need to define a libref explicitly to create a SAS file because SAS makes the association automatically. When you are creating a new SAS file and you specify a two-level SAS name, SAS automatically issues a FILEDEF using the libref (first-level name) for the DDname parameter **and** for the filetype unless a FILEDEF with that DDname is already in effect. By default, the data set will be written to your A-disk. For example, suppose you are creating a SAS data set called FOOD.PRICES:

```
DATA FOOD.PRICES;
  INPUT ... ;
more SAS statements
```

When SAS reads the name FOOD.PRICES, it checks to see if a FILEDEF with DDname FOOD has been issued. If there is no such FILEDEF, SAS issues one automatically, using FOOD for both the DDname and filetype, PRICES as the filename, and A as the filemode:

```
FILEDEF FOOD DISK PRICES FOOD A
```

Note: do not use the same filetype for SAS files and external (non-SAS) files. If you want a new SAS data set to be stored on a disk other than your A-disk, you must issue the appropriate FILEDEF explicitly. Specify the appropriate values for the DDname and filemode parameters. You can specify any value for filename and filetype; SAS will substitute the libref (first-level name) and data set name (second-level name) for filetype and filename, respectively. For example, if you want to store FOOD.PRICES on a B-disk, your FILEDEF could be:

```
FILEDEF FOOD DISK DUMMY DUMMY B
```

Notice that CMS and SAS identify the file by the same two names, but the order is reversed. For example, SAS calls the data set FOOD.PRICES, and CMS calls it PRICES FOOD.

Creating an External File

To create an external file in a SAS program, first issue a FILEDEF command to associate a fileref with the name of the external file to be created. Then use the fileref in the FILE statement of your SAS program. For example, suppose you write a DATA step that reads the external disk file CLIMATE DATA (which is stored on your A-disk) and creates a second external file called COLD TEMPS A. You use the filerefs DAILY and FREEZE to reference the files:

```
R;
FILEDEF DAILY DISK CLIMATE DATA A
R;
FILEDEF FREEZE DISK COLD TEMPS A
R;
SAS
1?
DATA _NULL_;
2?
INFILE DAILY;
3?
INPUT TEMP PRECIP SUNHRS;
4?
FILE FREEZE;
5?
more SAS statements
```

DETAILS FOR THE VMS OPERATING SYSTEM

The SAS System running under the VMS operating system is documented in "Changes and Enhancements in the Base SAS and SAS/GRAPH Products under VMS," SAS Technical Report P-128.

Invoking the SAS System

Enter the SAS command after you receive the $ prompt:

```
$ SAS
```

or

```
$ SAS/options
```

or

```
$ SAS filename
```

If you enter *SAS* you will invoke the SAS System in line-prompt mode. Use the second method to specify valid *options* when you invoke SAS. For example, you can specify *FSDEVICE=devicename* to invoke SAS in display manager mode. With the third method, you can specify *filename* to give the name of a file containing the SAS program to be executed. This method is called *noninteractive mode*.

Read more about the SAS command and how to execute SAS in batch mode in Technical Report P-128.

Defining Files

The method that you use to define a file depends on whether the file is a SAS or non-SAS (external) file. The method that you use allows the SAS System to distinguish between SAS files and external files.

External files You can define an external file in one of the following two ways:

- with the ASSIGN command of the DCL control language before you invoke the SAS System
- in the FILENAME statement within you SAS program.

With either method you associate the *fileref* with a complete file name, which includes the directory name and the name of a specific file within the directory. You can also issue the ASSIGN command in the X statement, a SAS statement that allows you to enter DCL commands in your SAS program.

The fileref is then used in SAS statements, such as INFILE and FILE, to refer to external files. If you specify a fileref in a SAS statement before you define the external file, the SAS System reads or creates a file in the current default directory with the following name:

```
[currentdefaultdirectory]fileref.extension
```

where *fileref* is specified in the INFILE statement to give the name of the file that is read from the current default directory or where *fileref* is specified in the FILE statement to give the name of the file being created in the current default directory.

Extension is a part of the file name that indicates the type of information a file contains. You must include the extension in the complete file name when you define an external file. (Refer to SAS Technical Report P-128 for more information on extension names.)

SAS files If you want to read or create a permanent SAS file, you must define the appropriate directory with the LIBNAME statement. In the LIBNAME statement you associate a *libref* with a directory name only. The libref is then used in SAS statements, such as DATA, SET, MERGE, and so on, to refer to the directory. Once you have defined a directory in the LIBNAME statement, you can use the associated libref to refer to any file in that directory.

If you do not include the LIBNAME statement in your program before a SAS statement that specifies the libref, you will get and error message stating that the directory to which you are referring cannot be found.

Note: discussion of SAS files in the following examples is limited to the most commonly used type, a **SAS data set**. A SAS data set is a SAS file containing observations and variables.

Methods for defining SAS data sets and external files are illustrated in the following sections.

Special Naming Conventions

Although the SAS System allows you to use an underscore (_) in SAS names, VMS does not. Therefore, do not use the underscore character in a libref or a fileref.

Reading an External File

You can use the FILENAME statement to define an external file. In the following program the FILENAME statement associates a fileref with a complete file name:

```
SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY '[YOURDIR1]YOURRAW.DAT';
2? DATA WEATHER;
3?    INFILE DAILY;
4?    INPUT TEMP PRECIP SUNHRS;
more SAS statements and messages
7? ENDSAS;
```

The FILENAME statement associates the fileref DAILY with the complete file name [YOURDIR1]YOURRAW.DAT. DAILY is then specified in the INFILE statement. Notice that the file name includes the directory name YOURDIR1 and the file YOURRAW within the directory DAT is the extension name.

You can also use the ASSIGN command to define the permanent external file before you invoke SAS. For example,

```
ASSIGN [YOURDIR1]YOURAW.DAT DAILY
```

Notice that the fileref DAILY is specified after the complete file name in the ASSIGN statement, which is opposite from the way you specify the FILENAME statement. If you chose to define the file by issuing the ASSIGN command with the X statement, you can simply replace the FILENAME statement in the above program with the following SAS statement:

```
X 'ASSIGN [YOURDIR1]YOURRAW.DAT DAILY';
```

Since the X statement is a SAS statement, be sure to put a semicolon at the end.

If you do not define the external file to be read by the above program, the SAS System will read (or attempt to read) a file from the current default directory with the following name:

```
[currentdefaultdirectory]DAILY.DAT
```

Reading a SAS Data Set

Use the LIBNAME statement to define the directory containing the SAS data set to be read, as shown in the following program:

```
SAS (invoke the SAS System)
(SAS messages)
1?' LIBNAME REPORT '[DIR2]';
2? PROC PRINT DATA=REPORT.STATION;
3?    VAR A B C;
```

```
more SAS messages
4? ENDSAS;
```

The LIBNAME statement associates the libref REPORT with the directory [DIR2]. REPORT is then used as the first level of the permanent SAS data set name in the PROC PRINT statement. When this program executes, the SAS System reads a SAS data set named STATION from the directory named [DIR2], and PROC PRINT prints the values for the variables A, B, and C.

If you specify a libref that has not been associated with a directory in the LIBNAME statement, you will receive an error message that the directory cannot be found.

Creating a SAS Data Set

The following program creates a SAS data set using an external file as input. You must use the LIBNAME statement to define the directory in which to store the SAS data set. You must also use the FILENAME statement to associate a fileref with the complete file name [YOURDIR1]YOURRAW.DAT.

```
SAS (invoke the SAS System)
SAS messages
1? FILENAME DAILY '[YOURDIR1]YOURRAW.DAT';
2? LIBNAME MAP '[YOURDIR2]';
3? DATA MAP.WEATHER;
4?    INFILE DAILY;
more SAS statements and messages
5? ENDSAS;
```

The LIBNAME statement associates the libref MAP with the directory [YOURDIR2], the directory in which WEATHER is stored. MAP is then specified as the first-level name of the SAS data set in the DATA statement.

If the LIBNAME statement is not specified before a statement that specifies the libref, you will receive an error message that the directory cannot be found. If you do not specify the FILENAME statement, the SAS System will search the current default directory for a file named DAILY.DAT.

Creating an External File

When you are creating an external file, you can use the FILENAME statement or the ASSIGN command to define the complete file name. The following program uses a SAS data set as input to create an external file:

```
SAS (invoke the SAS System)
SAS messages
1? FILENAME FREEZE '[DIR3]COLDRAW.DAT';
2? LIBNAME MAP '[YOURDIR2]';
3? DATA _NULL_;
4?    SET MAP.WEATHER;
5?    FILE FREEZE;
6?    IF TEMP < 32 THEN PUT TEMP;
more SAS statements and messages
9? ENDSAS;
```

The FILENAME statement associates the fileref FREEZE with the complete file name [DIR3]COLDRAW.DAT, and the LIBNAME statement associates the libref MAP with the directory containing WEATHER, the SAS data set being used as input.

When this program executes, all TEMP values less than 32 are written to an external file named COLDRAW.DAT. This file is located in the directory [DIR3]. If you do not specify the FILENAME statement, the SAS System writes the values in a file in the current default directory. The file has the following name:

```
[currentdefaultdirectory]FREEZE.DAT
```

DETAILS FOR THE VSE OPERATING SYSTEM AND ICCF

The *SAS Companion for the VSE Operating System* and "Enhancements and Updates for the VSE Operating System," SAS Technical Report P-132, give complete information on using the SAS System with the VSE operating system and ICCF.

The examples in this section use the word SAS to invoke the SAS System.

VSE Batch

Invoking the SAS System Use the EXEC statement in your JCL:

```
// EXEC PROC=SAS
```

In this case, a VSE batch procedure named *SAS* contains all the standard job control language required to invoke the SAS System in batch mode. Refer to the *SAS Companion for the VSE Operating System* for information on how to specify options at SAS invocation.

If your installation does not support a batch procedure to invoke the SAS System, your SAS consultant can tell you how to invoke SAS.

Defining files To define files stored on disk use the following set of JCL statements: DLBL, EXTENT, and ASSGN. You must issue this set of statements for each disk file needed—SAS and non-SAS. (In this context non-SAS means an external file—a file that is not a SAS file.)

The DLBL statement associates a *fileref* with an external file or a *libref* with a library of SAS files. In VSE terminology the fileref or the libref is called a *filename*. The EXTENT statement provides information about disk space occupied by the file, names a disk volume, and assigns a logical unit. The ASSGN statement associates the logical unit with the disk volume.

For example, the following statements define an external file stored on disk:

```
// DLBL fileref,'external.file.name',expiry
// EXTENT logicalunit,volumeserial,type,seq,begin,trks|blks
// ASSGN logicalunit,DISK,VOL=volumeserial,disposition
```

Use the following statements to define a library of SAS files stored on disk:

```
// DLBL libref,'SAS.library.name',expiry
// EXTENT logicalunit,volumeserial,type,seq,begin,trks|blks
// ASSGN logicalunit,DISK,VOL=volumeserial,disposition
```

The *fileref* is then used in SAS statements, such as FILE and INFILE, to refer to a permanent external file, and the *libref* is then used in SAS statements, such as DATA, SET, MERGE, and so on, to refer to a permanent SAS library. Once defined the libref can be used repeatedly in SAS statements to read or create SAS files in the library.

These three statements must be specified in the order shown above. Notice that both DLBL statements show how to specify an expiration date (expiry), and the EXTENT statements show how to specify full extent information (type,seq,

begin,trks | blks). These specifications are required when you are creating either a new SAS library or an external file, but they are not necessary for existing files. Example programs in the following sections use existing files.

Note: a SAS library can contain several different types of files. This discussion of SAS files is limited to one type, **SAS data sets**. A SAS data set is a SAS file containing observations and variables.

To define files stored on tape, use the TLBL, ASSGN, PAUSE, MTC, and UPSI statements. These statements are explained in the *SAS Companion for the VSE Operating System*, along with more detailed information on how to define tape and disk files.

If you use a libref or fileref in a SAS statement without having defined the file to which you are referring, you will receive an error message.

Special naming conventions The following information concerning fileref and libref naming conventions is summarized from the *SAS Companion for the VSE Operating System*:

- **Filerefs** or **librefs** specified in DLBL and TLBL statements must not exceed seven characters, although most SAS documentation states that as many as eight characters are allowed.
- The first letter of the **libref** specifies how the SAS library is accessed, that is, how the file is opened and whether you can write to it. Use one of the following letters as the first letter in a libref to indicate the appropriate access mode:

  W to specify a work library assumed to be a temporary file.

  O to create a new SAS library, not to be confused with writing a newly created SAS data set to an existing SAS library. The EXTENT statement must contain full extent information, as shown above.

  I or S to read a SAS file from a SAS library; the file can be used for input only.

  U (or any letter except W, O, S, or I) to specify read and write access to a SAS library.

- To access a permanent external file or SAS library stored on tape, you must use a libref or fileref ending with a 1-, 2-, or 3-digit number that matches the logical unit number specified in the control language.

These naming conventions are reflected in the following examples where appropriate. For more information on VSE naming conventions, please refer to the *SAS Companion for the VSE Operating System* and "Enhancements and Updates in SAS82.4 under VSE," SAS Technical Report P-132.

Methods for defining SAS libraries and external files are illustrated in the following sections.

Reading an external file In the following example, you must use the DLBL, EXTENT, and ASSGN statements to define the external file to be read:

```
* $$ JNM=jobname ...
* $$ LST LST= ...
* $$ LST LST= ...
// JOB jobname ...
// DLBL DAILY,'YOUR.RAW.DATA'
// EXTENT SYS050,VSE123
// ASSGN SYS050,DISK,VOL=VSE123,SHR
```

```
// EXEC  PROC=SAS
DATA WEATHER;
   INFILE DAILY RECFM=FB BLKSIZE=192 LRECL=1920;
   INPUT TEMP PRECIP SUNHRS;
more SAS statements
```

The DLBL statement associates the fileref DAILY with the file named YOUR.RAW.DATA. DAILY is then used in the INFILE statement

Note: the INFILE statement, in addition to giving fileref DAILY, must also supply the record format and block size. You must specify these two file characteristics for any file that you are reading by using the SAS options RECFM= and BLKSIZE=. You must also give the logical record length (LRECL=) if the file's record format is FB (fixed blocked) or VB (variable blocked).

Reading a SAS data set Use DLBL, EXTENT, and ASSGN statements to define a SAS library. In the following job, the DLBL statement associates the libref REPORT with a SAS library named YOUR.SAS.LIB. REPORT is then used in the PROC PRINT statement as the first level of the permanent SAS data set name:

```
* $$ JNM=jobname ...
* $$ LST LST= ...
* $$ LST LST= ...
// JOB jobname ...
// DLBL REPORT,'YOUR.SAS.LIB'
// EXTENT SYS050,VSE123
// ASSGN SYS050,DISK,VOL=VSE123,SHR
// EXEC  PROC=SAS
PROC PRINT DATA=REPORT.STATION;
   VAR A B C;
more SAS statements
```

When this program executes, the SAS data set named STATION is read from the SAS library YOUR.SAS.LIB, and PROC PRINT prints all the values for variables A, B, and C.

Note: if you specify a libref beginning with the letter I or S in this example, you have read-only access to the SAS library. For example, if YOUR.SAS.LIB is protected and allows only read access, you must specify IREPORT instead of REPORT. Remember that a libref beginning with a letter other than W, I, S, or O requests read and write access.

Creating a SAS data set The following program creates a permanent SAS data set using an external file as input. Therefore, you must use two sets of DLBL, EXTENT, and ASSGN statements: one set to define the SAS library that will store the newly created SAS data set and one to define the external file to be read.

```
* $$ JNM=jobname ...
* $$ LST LST= ...
* $$ LST LST= ...
// JOB jobname ...
// DLBL DAILY,'YOUR.RAW.DATA'
// EXTENT SYS050,VSE123
// ASSGN SYS050,DISK,VOL=VSE123,SHR
// DLBL MAP,'YOUR.SAS.LIB'
// EXTENT SYS067,ABC321
// ASSGN SYS067,DISK,VOL=ABC321,SHR
// EXEC  PROC=SAS
DATA MAP.WEATHER;
```

```
   INFILE DAILY RECFM=recfmBLKSIZE=blksizeLRECL=lrecl;
   INPUT TEMP PRECIP SUNHRS;
more SAS statements
```

The first DLBL statement associates the fileref DAILY with the permanent external file YOUR.RAW.DATA. DAILY is then used in the INFILE statement. The second DLBL statement associates the libref MAP with the SAS library YOUR.SAS.LIB. MAP is then used as the first level of the permanent SAS data set name in the DATA statement. When this program executes the new SAS data set WEATHER is stored in YOUR.SAS.LIB. Observations in WEATHER will contain the variables TEMP, PRECIP, and SUNHRS.

Creating an external file The following program creates an external file using a SAS data set as input. Again, you must include one set of JCL statements to define the SAS library to be used as input and one set to define the external file that is created by this program.

```
* $$ JNM=jobname ...
* $$ LST LST= ...
* $$ LST LST= ...
// JOB jobname ...
// DLBL MAP,'YOUR.SAS.LIB'
// EXTENT SYS051,VSE123
// ASSGN SYS051,DISK,VOL=VSE123,SHR
// DLBL FREEZE,'COLD.RAW.DATA',99/365
// EXTENT SYS050,VSE123,1,0,5280,300
// ASSGN SYS050,DISK,VOL=VSE123,SHR
// EXEC  PROC=SAS
DATA _NULL_;
   SET MAP.WEATHER;
   FILE FREEZE;
   IF TEMP < 32 THEN PUT TEMP;
   more SAS statements
```

The first DLBL statement associates the libref MAP with the SAS library YOUR.SAS.LIB. MAP is then used in the SET statement as the first level of the permanent SAS data set name. The second DLBL statement associates the fileref FREEZE with the permanent external file COLD.RAW.DATA. FREEZE is then used in the FILE statement. When this program executes, all TEMP values less then 32 that are read from the SAS data set WEATHER are written to the file COLD.RAW.DATA. These values replace existing information in the file.

ICCF (VSE Interactive)

Invoking the SAS System Enter the ICCF proc (short for procedure) after you receive the *READY prompt:

```
*READY
SAS
```

The ICCF proc (named *SAS*, in this case) contains the standard job control language used to invoke the SAS System. If you are using a full-screen terminal, display manager mode is default; otherwise, line-prompt mode is the default. Refer to the *SAS Companion for the VSE Operating System* for information on how to specify options at SAS invocation.

If your installation does not support a proc to invoke the SAS System, your SAS consultant can tell you how to invoke SAS.

Defining files In the ICCF proc that is used to invoke the SAS System, you must include a /FILE statement to define each permanent file—SAS and non-SAS. In this context non-SAS means an external file—any file that is not a SAS file.

In the /FILE statement you associate a *fileref* with a permanent external file and a *libref* with a library of SAS files. In ICCF terminology the fileref or the libref is called a *filename*. For example, to define an external file include the /FILE statement in the ICCF proc:

```
/FILE NAME=fileref,ID'external.file.name',UNITS=logicalunit,
/  SERIAL=volumeserial,DATE=expiry,LOC=begin,trks|blks
```

The *fileref* is then used in SAS statements, such as FILE and INFILE, to refer to the external file.

To define a SAS data library, include the following /FILE statement in the ICCF proc:

```
/FILE NAME=libref,ID'SAS.library.name',UNITS=logicalunit,
/  SERIAL=volumeserial,DATE=expiry,LOC=begin,trks|blks
```

The *libref* is then used in SAS statements, such as DATA, SET, MERGE, and so on, to refer to the library of SAS files.

The /FILE statements shown above include parameters for specifying an expiration date (DATE=expiry), beginning tracks, and number of tracks or blocks used (LOC=begin,trks | blks). This information is required if you are creating a new SAS library or a new external file, but it is not necessary if you are using an existing library.

Note: UNITS must specify a logical unit that was assigned to the disk volume *volumeserial* in the ICCF start-up JCL. See your systems personnel for more information on logical units.

You cannot use tape files with the SAS System running under ICCF.

If you use a libref or fileref in a SAS statement without having defined the file to which you are referring, you will receive an error message.

Note: a SAS library can contain several different types of files. This discussion of SAS files is limited the most commmonly used type, a **SAS data set**. A SAS data set is a SAS file containing observations and variables.

Special naming conventions The following information concerning fileref and libref naming conventions is summarized from the *SAS Companion for the VSE Operating System*:

- Filerefs or librefs specified in /FILE statements must not exceed seven characters, although most SAS documentation states that as many as eight characters are allowed.
- The first letter of the **libref** specifies how the SAS library is accessed, that is, how the file is opened and whether you can write to it. Use the following letters as the first letter in a libref to indicate the appropriate access mode:

W	to specify a work library assumed to be a temporary file.
O	to create a new SAS library, not to be confused with writing a newly created SAS data set to an existing SAS library. Remember to include the DATE= and LOC= parameters in the /FILE statement, as shown above.
I or S	to read a SAS file from a SAS library; the file is used for input only.

U (or any letter except W, O, S, or I) to specify read and write access to a SAS library.

These naming conventions are reflected in the following examples where appropriate. For more information on VSE naming conventions, please refer to the *SAS Companion for the VSE Operating System* and "Enhancements and Updates in SAS82.4 under VSE," SAS Technical Report P-132.

Methods for defining SAS libraries and external files are illustrated in the following sections.

Reading an external file If your SAS program reads an external file, you must define that file by including the /FILE statement in the ICCF proc. Assume that the file to be read is an existing file named YOUR.RAW.DATA. Include the following statement:

```
/FILE NAME=DAILY,ID'YOUR.RAW.DATA',UNITS=SYS050,SERIAL=VSE123
```

The /FILE statement associates the fileref DAILY with the external file YOUR.RAW.DATA, an existing file.

Now, invoke the SAS System and enter your program. You will use the fileref DAILY in the INFILE statement to refer to the external file:

```
SAS (invoke the SAS System)
SAS messages
*ENTER DATA?
DATA WEATHER;
*ENTER DATA?
INFILE DAILY RECFM=recfm BLKSIZE=blksize LRECL=lrecl;
*ENTER DATA?
INPUT TEMP PRECIP SUNHRS;
more SAS statements and messages
*ENTER DATA?
ENDSAS;
```

Note: the INFILE statement, in addition to fileref DAILY, must also supply the record format and block size. You must specify these two file characteristics for any file that you are reading by using the SAS options RECFM= and BLKSIZE=. You must also give the logical record length (LRECL=) if the file's record format is FB (fixed blocked) or VB (variable blocked).

Reading a SAS data set The following program reads a SAS data set from an existing SAS library. You must define the library by including a /FILE statement in the ICCF proc:

```
/FILE NAME=REPORT,ID'YOUR.SAS.LIB',UNITS=SYS050,SERIAL=VSE123
```

The /FILE statement associates the libref REPORT with the SAS library named YOUR.SAS.LIB.

You are now ready to invoke the SAS System and enter the program:

```
SAS (invoke the SAS System)
SAS messages
*ENTER DATA?
PROC PRINT DATA=REPORT.STATION;
*ENTER DATA?
VAR A B C;
      more SAS messages
*ENTER DATA?
ENDSAS;
```

The libref REPORT is used in the PROC PRINT statement as the first level of the permanent SAS data set name. When this program executes the SAS data set named STATION is read from the SAS library YOUR.SAS.LIB, and PROC PRINT prints all values for variables A, B, and C.

Note: if you specify a libref beginning with the letter I or S, you have read-only access to the SAS library. For example, if you specify IREPORT instead of REPORT in this example, you have read-only access to YOUR.SAS.LIB. Remember that a libref beginning with a letter other than W, I, S, or O requests read and write access.

Creating a SAS data set The following program creates a SAS data set using an external file as input. Therefore, you include two /FILE statements in the ICCF proc: one to define the SAS library that will store the newly created SAS data set and one to define the external file to be used as input:

```
/FILE NAME=DAILY,ID'YOUR.RAW.DATA',UNITS=SYS050,SERIAL=VSE123
/FILE NAME=MAP,ID'YOUR.SAS.LIB',UNITS=SYS067,SERIAL=ABC321
```

The first /FILE statement associates the fileref DAILY with the permanent external file YOUR.RAW.DATA. The second /FILE statement associates the fileref MAP with the permanent SAS data library YOUR.SAS.LIB, an existing SAS library.

You are now ready to invoke the SAS System and enter the program statements:

```
SAS (invoke the SAS System)
SAS messages
*ENTER DATA?
DATA MAP.WEATHER;
*ENTER DATA?
INFILE DAILY RECFM=recfm BLKSIZE=blksize LRECL=lrecl;
*ENTER DATA?
INPUT TEMP PRECIP SUNHRS;
more SAS statements and messages
*ENTER DATA?
ENDSAS;
```

DAILY is used in the INFILE statement to refer to the external file. MAP is used as the first level of the permanent SAS data set name in the DATA statement. When this program executes, it creates the new SAS data set WEATHER in YOUR.SAS.LIB. Observations in WEATHER will contain the variables TEMP, PRECIP, and SUNHRS.

Creating an external file The following program creates an external file using a SAS data set as input. Again, in the ICCF proc you must include one /FILE statement to define the SAS library to be used as input and another /FILE statement to define the external file that is created by this program:

```
/FILE NAME=MAP,ID'YOUR.SAS.LIB',UNITS=SAS051,SERIAL=VSE123
/FILE NAME=FREEZE,ID'COLD.RAW.DATA',UNITS=SYS050,SERIAL=VSE789
```

The first /FILE statement associates the libref MAP with an existing SAS library, YOUR.SAS.LIB. The second /FILE statement associates the fileref FREEZE with the external file COLD.RAW.DATA, also existing.

You are now ready to invoke the SAS System and enter the program:

```
SAS (invoke the SAS System)
SAS messages
*ENTER DATA?
DATA _NULL_;
*ENTER DATA?
```

```
SET MAP.WEATHER;
*ENTER DATA?
FILE FREEZE;
*ENTER DATA?
IF TEMP < 32 THEN PUT TEMP;
more SAS statements and messages
*ENTER DATA?
8? ENDSAS;
```

The libref MAP is used as the first level of the permanent SAS data set name in the SET statement. The fileref FREEZE is used in the FILE statement. When this program executes, all TEMP values less then 32 that are read from the SAS data set WEATHER are written to the file COLD.RAW.DATA. These values replace existing information in the external file.

NOTES

1. **AOS/VS, PRIMOS, VMS**: If you do not define the external file to be read or created by your SAS program, the SAS System uses a file in the current default directory with the following name:

AOS/VS	*:UDD:currentdefaultdirectory:fileref*
PRIMOS	*<masterdirectory>currentdirectory>fileref*
VMS	*[currentdefaultdirectory]fileref.DAT*

2. **AOS/VS, PRIMOS, VMS**: A SAS data library is a logical concept, not a physical entity. All permanent SAS files in a directory belong to the same SAS data library.

CMS, VM/PC: A SAS data library is a logical concept, not a physical entity. All SAS files with the same filetype and filemode make up a logical SAS data library.

OS, VSE: A SAS data library is a physical data set— an OS or VSE data set that contains one or more SAS files.

3. **AOS/VS, PRIMOS, VMS**: The LIBNAME statement is available if you are running the SAS System under one of these operating systems.

4. **AOS/VS, PRIMOS, VMS**: The LIBSEARCH statement is available if you are running the SAS System under one of these operating systems.

5. **CMS, OS**: Under TSO you can use the SAS TSO statement to issue TSO commands from within your SAS program. Under CMS you can use the SAS CMS statement to issue CMS commands from within your SAS program. The X statement is used like a CMS statement under CMS and like a TSO statement under TSO.

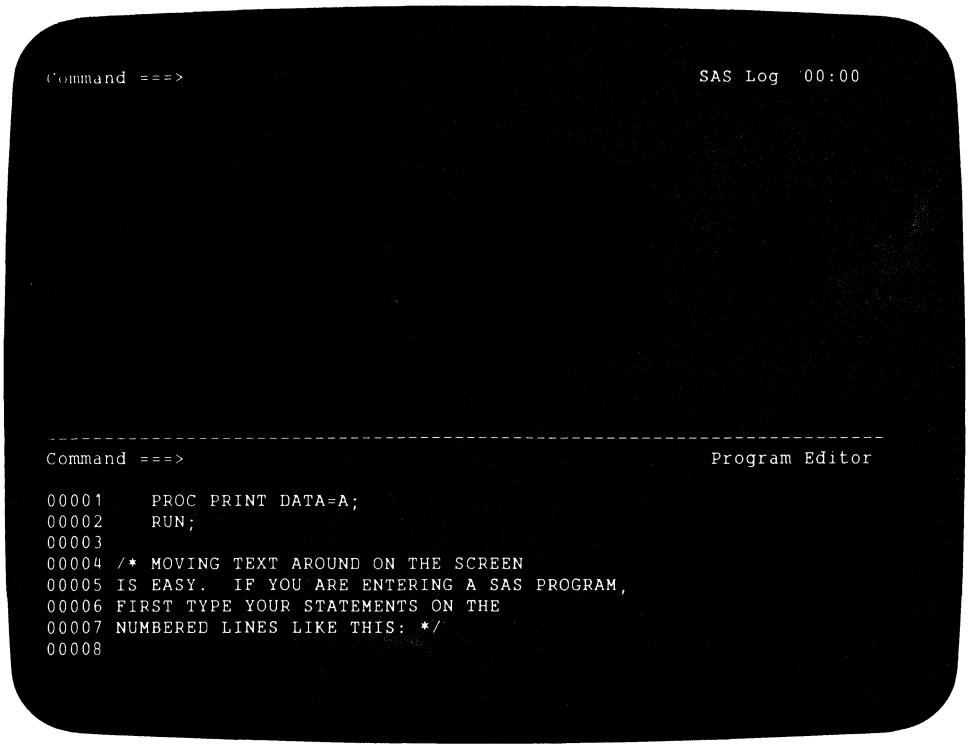
Appendix 3

Full-Screen Editing

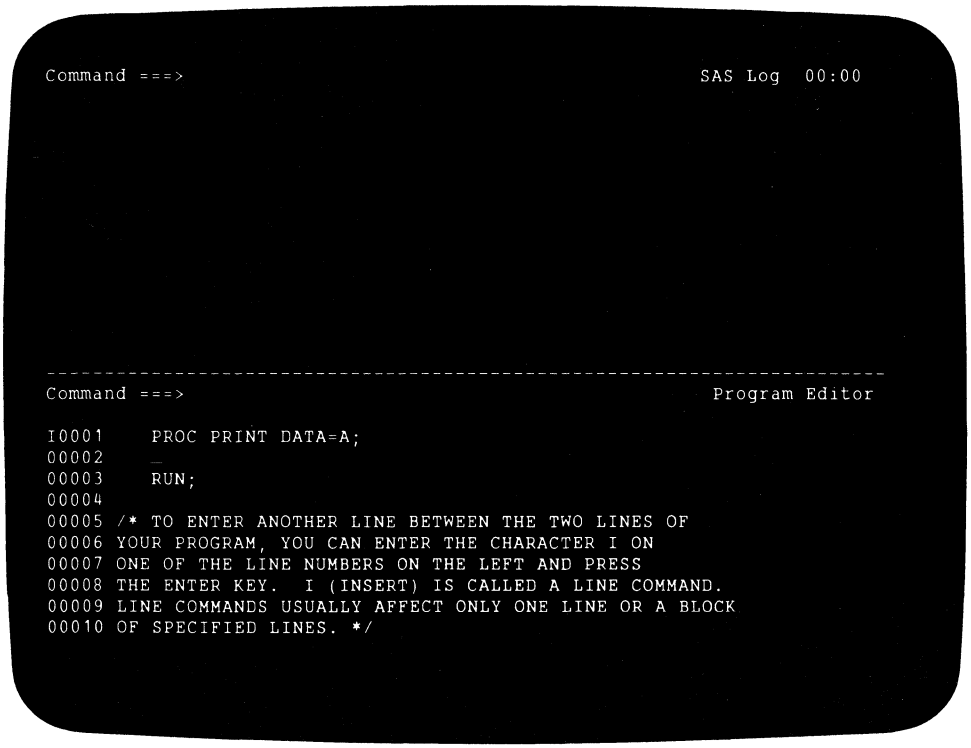
INTRODUCTION

This appendix shows you how to enter information if you are using the SAS System with a full-screen terminal. Use the editing features described here with the SAS Display Manager System for entering SAS program statements. These editing features can also be used to enter text in other full-screen SAS software products, for example, to enter FSLETTER text in the SAS/FSP software product or to compose screen applications with the SAS/AF software product. You can also set your terminal keys to convenient functions with the SAS user profile facility.

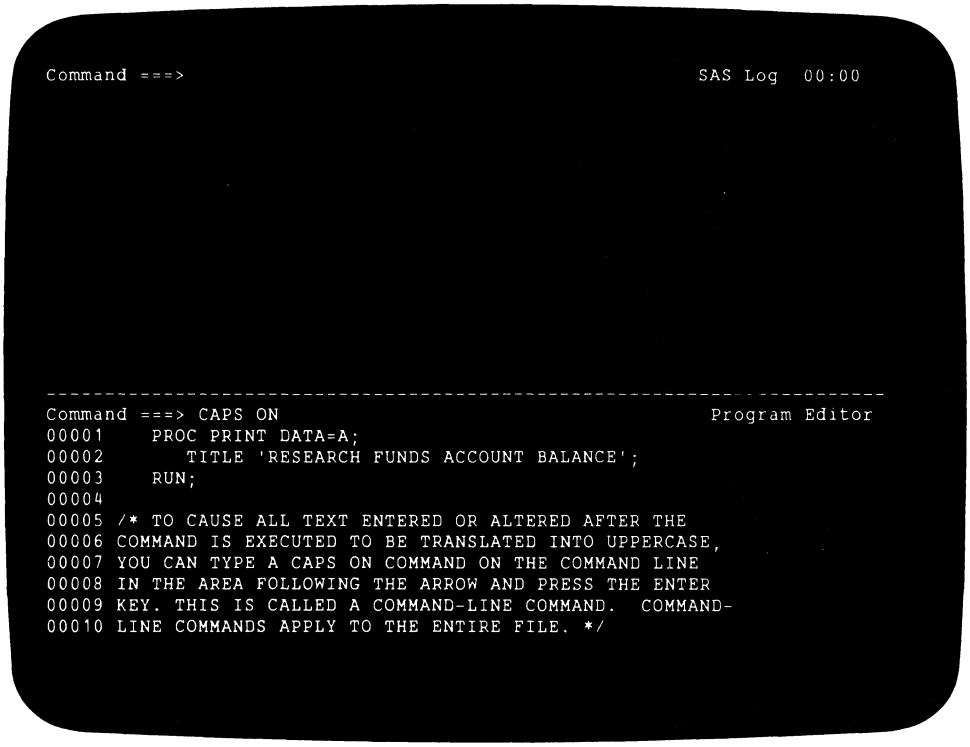
The display manager examples below illustrate full-screen editing capabilities that are available wherever you need them in the SAS System.



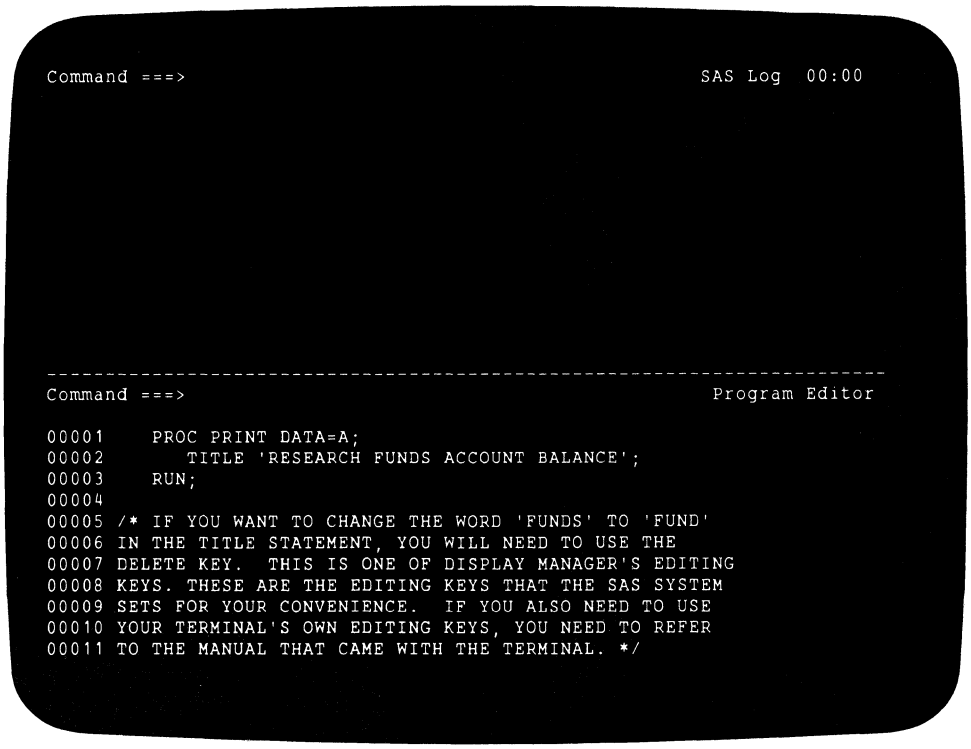
Screen A3.1 Entering Text



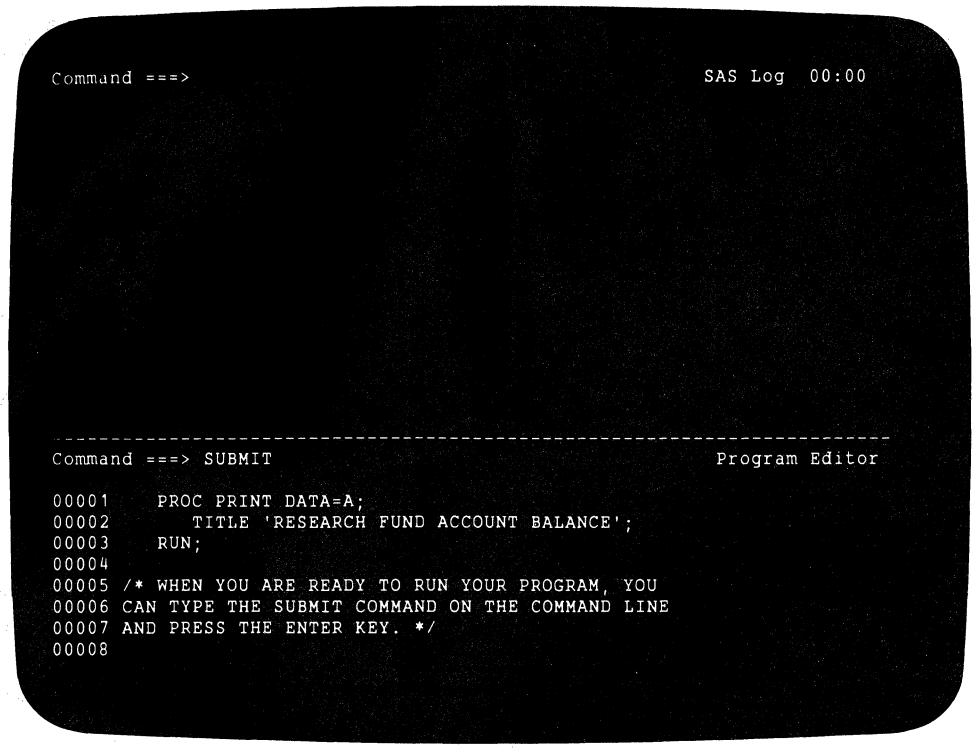
Screen A3.2 Inserting Lines with a Line Command



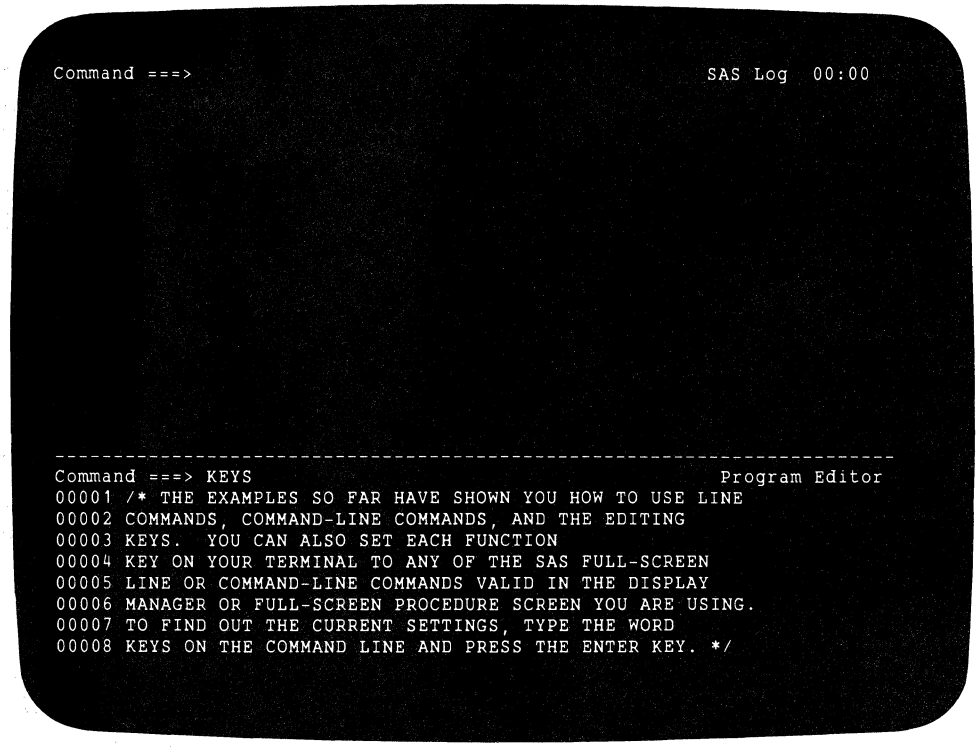
Screen A3.3 Translating Text into Uppercase with a Command-Line Command



Screen A3.4 Deleting Characters with an Editing Key



Screen A3.5 Submitting SAS Program Statements



Screen A3.6 Function Key Settings

The result of the KEYS command is shown below. These are the default function key settings for the program editor screen that SAS Institute provides to your site; yours may be different from those shown.

```
            FUNCTION KEY DEFINITIONS
Command ===>
KEY     COMMAND
01      help
02      split
03      submit
04      recall
05      rfind
06      rchange
07      backward
08      forward
09      output
10      left
11      right
12      cursor
13      help
14      split
15      submit
16      recall
17      rfind
18      rchange
19      backward
20      forward
21      output
22      left
23      right
24      cursor
```

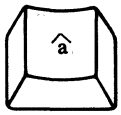
Screen A3.7 Function Key Definition Screen

Notice that one of the function keys is set to execute the SUBMIT command. Instead of typing the word SUBMIT on the command line and pressing ENTER as you did above to submit your program statements, simply press the function key set to execute the SUBMIT command.

EDITING KEYS

The editing keys you can use with display manager and the procedures that allow full-screen editing are shown below. You can use these same keys when editing text on data lines and commands on command lines. In general, you use an editing key to make a change to one character or one line of an unprotected field and a function key to search or scroll an entire file. The editing keys you can use in full-screen editing are listed and defined in **Figure A3.1**.

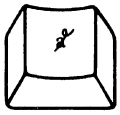
Note: the words or symbols on the editing keys described in **Figure A3.1** may not be the same as those on the terminal you are using.[1]



The INSERT key allows you to insert characters within a line. Press the key and position the cursor where you want to insert text. Characters are shifted to the right to make room for new text. *

---

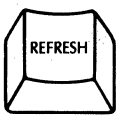
* **IBM users**: On a 3270 series terminal, press the RESET key to discontinue the insert.
Minicomputer users: Press the INSERT key again to discontinue the INSERT.



The DELETE key erases the character either preceding the cursor position or in the current cursor position, depending on the terminal keyboard you are using.

The ERASE EOF (End-of-Field) key deletes or erases all the characters from the cursor position to the end of either a data entry line or, on some screens, the end of a field.

The HOME key moves the cursor to the first column of the first unprotected field on the screen, the entry area of the command line on most screens. *

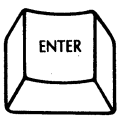


The REFRESH key redisplays the contents of the screen line-by-line. It removes messages from the operating system and is especially useful for removing unwanted characters input since the last time you pressed ENTER or a function key. **

The NEW LINE key moves the cursor to the first unprotected field of the next line. Use the NEXT FIELD key to move the cursor from the line number to the data entry area.

The PRIOR FIELD key moves the cursor to the previous unprotected field.

The NEXT FIELD key moves the cursor to the next unprotected field.



The ENTER key executes commands entered on a command line or on a line number.

Figure A3.1 Editing Keys

---

* **IBM users**: Press the ALT and Home keys.

** **CMS, OS, and VM/PC**: Press ALT PA2.
VSE: Press ALT CLEAR.

COMMANDS

Command Conventions

SAS full-screen editing commands follow these conventions:

COMMAND OPTION | *option* [| **OPTION** | *option* | ...]

where

bold CAPS
: indicates a **KEYWORD**; use exactly the same spelling and form as shown.

lowercase italic
: indicates you supply the actual value.

vertical bar |
: means *or* use only one of the terms separated by vertical bars.

brackets [ ]
: indicate optional information or keywords.

three periods (...)
: mean that more than one of the terms preceding ... can be optionally specified.

For example, in the FIND command

FIND *characterstring* [**PREFIX** | **SUFFIX** | **WORD**]

FIND
: is the command **KEYWORD**.

characterstring
: is a user-defined option. You supply the value of the string of characters you want to locate. It is not in brackets so you know you are required to specify a character string.

[**PREFIX** | **SUFFIX** | **WORD**]
: are three system-defined option **KEYWORDS** that appear in bold uppercase letters, enclosed by brackets [ ], separated by vertical bars |; the bold uppercase letters indicate that each is entered exactly as written; the brackets indicate that these options are allowed but not required; the vertical bars indicate that you can specify only one of the three.

Line Commands

You can use line commands to perform such tasks as moving, copying, inserting, and deleting lines or blocks of lines, and designating target locations for moved, copied, and inserted lines. As an example, a list of line commands that you can use for full-screen editing in the text editor in SAS software products follows. See "SAS Display Manager" appendix for the line commands available in the display manager program editor.

SINGLE COMMANDS:

A,B
: target position, A (after) or B (before), of an M, MM, C, CC, or COPY command.

C	copy a line to the location indicated either by an A (after) or B (before) line command. To copy more than one line, see the CC line command below.
D[*n*]	delete one or more lines.
I[*n* | A[*n*] | B[*n*]]	insert one or more lines immediately following or preceding an I command.
M	move a line to the location indicated either by an A or B line command. To move more than one line, see the MM line command below.
MASK	enter the mask to be used for newly inserted lines.
O	a target position command that overlays the contents of one or more lines with moved or copied lines.
R[*n*]	repeat a line *n* times immediately following that line.
TF[*n*]	flow lines of text, deleting trailing blanks. You can follow TF with some number *n* to specify the right margin.
TS[*n*]	insert one or more blank lines within a line of text.

BLOCK COMMANDS:

CC	copy block of lines designated by CC on the first and last lines.
DD	delete block of lines designated by DD on the first and last lines.
MM	move block of lines designated by MM on the first and last lines.
RR	repeat block of lines designated by RR on the first and last lines.

SPECIAL SHIFT COMMANDS:

> *n*	shift data *n* columns to the right.
< *n*	shift data *n* columns to the left.
> > *n*	shift data on block of lines *n* columns to the right.
< < *n*	shift data on block of lines *n* columns to the left.

Command-Line Commands

You can use command-line commands to perform many editing and file management tasks. As an example, the command-line commands that you can use in the SAS text editor are listed below by function:

Scrolling Commands

AUTOADD	LEFT
BACKWARD	*n*
FORWARD	RIGHT
TOP	HSCROLL
BOTTOM	VSCROLL

File Management Commands

COPY	END	INCLUDE

General Editing Commands

CHANGE	CAPS
RCHANGE	CPRO
DES	CUNPRO
FILL	CURSOR
FIND	NULLS
RFIND	NUMS
BOUNDS	PREVCMD
CANCEL	RESET

Help, Function Keys, and Host-level Commands

HELP	KEYS	X command

Many of these commands can also be used in other screens in SAS full-screen procedures, as well as in display manager. See "SAS Display Manager" appendix for a list and glossary of display manager commands.

USING COMMANDS: MORE EXAMPLES

Entering Commands Directly

Both command-line commands and line commands can be entered directly or executed with function keys. To execute a command with a function key, you need to know the function key settings. In some procedures you can alter function key settings and then permanently store the new settings. See the **Function Keys** section for a discussion of executing commands with function keys.

Entering command-line commands directly To enter a command-line command directly, type the command on the command line and press the ENTER key. For example, to scroll backward the default vertical scroll amount, type

```
BAC
```

on the command line and press ENTER. The BAC command is an example of the most basic kind of command to enter; it requires only that you type a keyword and press ENTER. The examples below include commands that allow options and commands that require cursor placement.

Command-line commands with options The BAC command allows options to be specified. For example, to scroll backward the maximum amount, type

```
BAC MAX
```

on the command line and press ENTER.

Entering line commands directly To enter a line command directly, type the command over a line number and press the ENTER key. In the text editor screens

of most full-screen procedures, unlike the program editor screen in display manager, the numbering facility is OFF by default. To number the data entry lines, use the NUMS ON command-line command. Type NUMS ON on the command line and press ENTER.

Examples below include single line commands, line commands with options, line commands requiring target commands and positioning of the cursor, and block commands. Note the underscore (_) marking the cursor position in several of the examples.

Single line command As an example, to insert a new line for entering text, type I, the insert line command, on any part of a line number

```
00I001 data line one
000002 data line two
000003 data line three
```

and press ENTER. One new line, the default, is inserted between the first and second lines.

```
000001 data line one
000002 _
000003 data line two
000004 data line three
```

Line command with option Some line commands, the I line command, for example, allow you to specify an option for which you supply the value, such as some number *n*. To specify how many lines to insert, follow the I line command with some number *n*, such as 3, and a blank space

```
I3 001 data line one
000002 data line two
000003 data line three
```

and press ENTER. Three new lines are inserted between the first and second lines.

```
000001 data line one
000002 _
000003
000004
000005 data line two
000006 data line three
```

Line command requiring cursor position The TS (text split) line command requires that you enter the command, then position the cursor before pressing ENTER. Enter the TS command on a line number, position the cursor,

```
TS0001 Split this line after the semicolon;_then add new copy
```

and then press ENTER. A new line is inserted:

```
000001 Split this line after the semicolon;_
000002
000003                                        then add new copy.
```

Line command requiring a target command The following example shows how to use the M (move) and A (after) line commands:

```
M00001 Move this line
A00002 after the second line
000003 with the M (move) and A (after) line commands.
```

The first line becomes the second line:

```
000001 after the second line
000002 Move this line
000003 with the M (move) and A (after) line commands.
```

Block line command You can also use line commands to affect blocks of lines. For example,

```
MM0001 Move the first two lines
MM0002 after the third line
A00003 with the M (move) and A (after) line commands.

000001 with the M (move) and A (after) line commands.
000002 Move the first two lines
000003 after the third line
```

See **Command Conventions** earlier in this appendix for a description of command conventions.

Executing line commands without line numbers If your data lines are not numbered, you can still execute line commands without having to use the NUMS ON command to display line numbers. Type the line command, preceded by a colon (:), on the command line, position the cursor on the line you want to be affected by the line command, and press ENTER. For example, to execute the TS (text split) line command from the command line, type :TS on the command line, position the cursor, and press ENTER.

You can also execute a line command with a function key even when line numbers are not displayed. Position the cursor where you want the command to take effect, and press a function key set to execute that line command. See **Altering Settings: the Function Key Definition Screen** for setting a function key to execute a line command.

Entering Multiple Commands

You can enter a series of commands on the command line by separating the commands with semicolons. For example, you can type the BACKWARD MAX scroll command and the FIND command on the command line, separate them with a semicolon,

```
BACKWARD MAX; FIND 'DATA ONE'
```

and execute both by pressing the ENTER key once.

FUNCTION KEYS

Executing Commands with Function Keys

To execute a command with a function key can be as simple as pressing the function key instead of typing a command on the command line and pressing ENTER. To use a function key to execute a command with an option, you can either assign that function key the command with the option, such as BACKWARD MAX, or you can type the option MAX on the command line and press the BACKWARD function key. You can execute a command that requires cursor placement, such as SPLIT, by positioning the cursor before pressing the SPLIT function key.

Function keys and multiple commands You can also combine the use of function keys and the submission of multiple commands. For example, if you set a function key to execute the command and option BACKWARD MAX followed by a semicolon,

```
BACKWARD MAX;
```

then you can enter

```
FIND 'DATA ONE'
```

on the command line and execute both by pressing the key you set to execute BACKWARD MAX; . The procedure executes the BACKWARD MAX command first and then the FIND command.

Altering Settings: the Function Key Definition Screen

You can alter the function key settings to execute any commands that are valid for the particular screen in the SAS full-screen software product that you are using. Use the KEYS command to display the function key definition screen for the screen you are currently using. Twenty-four function keys with settings are listed although some terminals may have only twelve function keys.

You can assign any valid command-line command or line command to any function key on your terminal. Precede line commands with a colon (:). You can use editing keys such as EOF and DELETE, or simply type over a function key setting to alter it. On the function key definition screen, you can use the following commands:

```
BACKWARD
CANCEL
FORWARD
END
```

You can use any of these commands by pressing a function key set to execute it or by typing it on the command line and pressing the ENTER key. You can scroll backward and forward to view the entire list of key settings; you can cancel any changes made; you can use the END command, or a key assigned to execute the END command, to save your altered settings and return to the screen on which you executed the KEYS command. Altered function key settings remain in effect until you exit from the procedure or until you alter the settings again. The next time you enter the procedure, the original default settings will be in effect again.

To keep your altered function key settings from one session to another, you must store them in your own user profile catalog. At the beginning of each SAS session, use the appropriate operating system commands to make your user profile catalog available. See the sections **User Profile** and **Access to Your PROFILE Catalog**.

SAS USER PROFILE

Installation Profile

Each installation of the SAS System receives a special SAS data library that contains information used by SAS software to control various aspects of your SAS session. For example, in base SAS software, a catalog in the installation profile library contains the default function key settings for the SAS Display Manager System and PROC DATASETS; similiar information for additional SAS software prod-

ucts is contained in other catalogs within that same library. In some cases, these default settings can be tailored to your site by your SAS installation representative.

Keep in mind that any changes made to the installation profile library are universal for your site. That is, the default settings stored in the catalogs in this library are for everyone using the SAS System at your installation. If these default settings are not suitable for your own applications, you can set up your own user profile. See **User Profile** below.

User Profile

A user profile catalog is available for customizing default function key settings to meet individual needs and preferences. When looking for current function key settings, for example, the SAS System searches the user profile for function key settings stored under the appropriate names before it looks in a catalog in the installation profile library.

Your own user profile information is stored in either a permanent catalog named SASUSER.PROFILE or a temporary catalog named WORK.PROFILE.[2] SASUSER is a reserved libref and PROFILE is a reserved SAS catalog name. The PROFILE catalog stores the function key settings for a particular screen or procedure each time you enter the function key definition screen and save the settings. You create a function key definition screen in your user profile catalog each time you use the KEYS command to display a function key definition screen and then save that screen by exiting it with the END command. By default, any function key definition screen that you create in this way is automatically stored under the appropriate name in the PROFILE catalog. For example, in base SAS software, you can have different function key settings for each display manager screen and PROC DATASETS because the system assigns a reserved name to each function key definition screen when it stores it in the user profile catalog. When you are using a particular display manager screen or full-screen procedure, the system looks for your function key setting information stored under a particular name for that screen or procedure only.

The user profile catalog is intended to be a personal catalog and cannot be accessed by more than one SAS user. Normally, an individual's profile is associated with an individual's user id.[3] It should not be stored in a SAS data library that is shared by a number of users.

Access to Your PROFILE Catalog

The individual user profile information is stored in a catalog named PROFILE in a SAS data library.[4] You should have access to your own user profile. In some cases, the reserved libref and catalog name SASUSER.PROFILE is used in a command procedure and automatically accessed when you invoke the SAS System at your site. In other cases, you may need to make the catalog SASUSER.PROFILE available to your session each time you execute the SAS System.

If you do not make the catalog SASUSER.PROFILE available to your session, WORK.PROFILE is opened and any function key definition screens you create are stored there. As with SAS data sets placed in temporary storage, your function key definition screens placed in the catalog WORK.PROFILE are available only for that session.

The library referenced by SASUSER, unlike the library containing the profile information for the entire installation, does not exist until you create it. (You can, however, reference an existing SAS data library using the libref SASUSER.) You must use a host operating system command to create a file or data set to contain a SAS data library that is associated with the reserved libref SASUSER. The system

creates a PROFILE catalog automatically the first time you make a library referenced by SASUSER available to your session. Note that you must have write access to this SAS data library.

NOTES

1. If you are using one of the terminals listed below, use **Table A3.1** to determine the keys that have been assigned special editing functions in SAS full-screen procedures and display manager. See **Figure A3.1** for a description of the functions of these editing keys. If you are using a terminal other than one of those listed below that does not have keys matching the words or symbols shown in **Figure A3.1**, consult your SAS representative. If you are using an IBM terminal in the 3270-series, you can disregard this section.

Table A3.1 Full-Screen Editing Functions Assigned to Terminal Keys

Full-Screen Editing Key	PT45 Key	PST100 Key	PT25 Key
â (INSERT)	ichar or shift/del	insert or backspace	shift/del
ß (DELETE)	dchar or backspace	delete or del	backspace
ERASE EOF	line feed	erase	shift/f8
HOME	home	home	home
REFRESH	control/ clear	pf10	control/erase
NEW LINE	return or enter	return or enter	new line or enter
NEXT FIELD	f13	f2	shift/f5
PRIOR FIELD	f14	f3	shift/f6
ENTER	f3 or f15	f1 or pf13	f3 or shift/f7
MOVE CURSOR UP	↑	↑	↑
MOVE CURSOR DOWN	↓	↓	(not available)
MOVE CURSOR LEFT	←	←	←
MOVE CURSOR RIGHT	→	→	←

continued on next page

Table A3.1 *continued*

Full-Screen Editing Key	TEK4104 Key	VT100 Key	DASHER Key
â (INSERT)	backspace	backspace	C1
⌫ (DELETE)	rub out	delete	DEL
ERASE EOF	line feed	line feed	erase EOL
HOME	- (minus sign on numeric keypad)	- (minus sign on numeric keypad)	home
REFRESH	, (on numeric keypad)	, (on numeric keypad)	erase page
NEW LINE	return	return	new line or enter
NEXT FIELD	0 (on numeric keypad)	0 (on numeric keypad)	C4
PRIOR FIELD	. (on numeric keypad)	. (on numeric keypad)	C3
ENTER	enter	enter	CR
MOVE CURSOR UP	F1	↑	↑
MOVE CURSOR DOWN	F2	↓	↓
MOVE CURSOR LEFT	F3	←	←
MOVE CURSOR RIGHT	F4	→	→

2. **CMS and VM/PC**: The SAS System automatically creates SASUSER.PROFILE for you on your A-disk the first time you enter display manager or a SAS full-screen procedure. The SASUSER.PROFILE catalog remains on your A-disk for use in different SAS sessions. Disregard all references to explicitly making SASUSER .PROFILE available to your session.

3. **CMS and VM/PC**: The SASUSER.PROFILE catalog is stored on your own A-disk. This catalog is created automatically by the SAS System. You need not issue any CMS commands to use it.

4. **CMS and VM/PC**: Skip the section **Access to Your PROFILE Catalog**.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Alice T. Allen edited the *SAS® User's Guide: Basics, Version 5 Edition.*

Writers	Brenda C. Kalt Marian Saffer Stephenie P. Joyner Carol Linden Gretel Easter J. Robin Keller Carol W. Myrick Regina Luginbuhl Mary Ann Kaufman Larry Crum
Contributing Editors	Kathryn A. Council John P. Sall

Editorial Services Copyeditors, proofreaders, and coders provided editorial support under the direction of **Judith K. Whatley**.

Copyeditors	David D. Baggett Betty Fried
Proofreaders	Gigi Hassan James K. Hart Frances A. Kienzle
Manuals composition coders	Gail C. Freeman Blanche Weatherspoon Amy G. Power
Administrative staff	Barbara D. Johnson Toni P. Sherrill June F. Zglinski
Other assistance	Holly S. Whittle Robert Hastings Rebeccah Neff

Graphic Arts provided the text composition and production under the direction of **Carol M. Thompson**.

Programmers	Craig R. Sampson Pamela A. Troutman
Compositors	Arlene B. Drezek Sarah M. Richardson
Artist	Michael J. Pezzoni

Your Turn

If you have comments about SAS software or the *SAS User's Guide: Basics, Version 5 Edition*, please send us your ideas on a photocopy of this page. If you include your name and address, we will reply to you.

Please return to the Publications Division, SAS Institute Inc., SAS Circle, Box 8000, Cary, NC 27511-8000.

0-917382-65-X