Sample Text . . . Chris Olsen’s Teaching Elementary Statistics with JMP® demonstrates this powerful software, offering the latest research on “best practice” in teaching statistics and how JMP can facilitate it. Just as statistics is data in a context, this book presents JMP in a context: teaching statistics. Olsen includes numerous examples of interesting data and intersperses JMP techniques and statistical analyses with thoughts from the statistics education literature. Intended for high school-level as well as college-level instructors who use JMP in teaching elementary statistics, the book uniquely provides a wide variety of data sets that will be of interest to a broad range of teachers and students.

ART CARPENTER Sample Text . . . is Professor of Statistics at Grinnell College in Grinnell, Iowa, where he teaches elementary statistics and experimental design. A JMP® user since JMP 3, he earned his PhD in educational administration, with concentrations in measurement and evaluation, research methodology, and educational computing systems, and his MA in mathematics education, both from the University of Iowa. Chris is a member of the National Council of Teachers of Mathematics and the American Statistical Association.
# Contents

- About This Book  xvi
- Acknowledgments  xxv
- About the Author  xxvii

## Part 1  Data Preparation  1

### Chapter 1  Moving, Copying, Importing, and Exporting Data  3

1.1 LIBNAME Statement Engines  4
   - 1.1.1 Using Data Access Engines to Read and Write Data  5
   - 1.1.2 Using the Engine to View the Data  6
   - 1.1.3 Options Associated with the Engine  6
   - 1.1.4 Replacing EXCEL Sheets  7
   - 1.1.5 Recovering the Names of EXCEL Sheets  8

1.2 PROC IMPORT and EXPORT  9
   - 1.2.1 Using the Wizard to Build Sample Code  9
   - 1.2.2 Control through the Use of Options  9
   - 1.2.3 PROC IMPORT Data Source Statements  10
   - 1.2.4 Importing and Exporting CSV Files  12
   - 1.2.5 Preventing the Export of Blank Sheets  15
   - 1.2.6 Working with Named Ranges  16

1.3 DATA Step INPUT Statement  17
   - 1.3.1 Format Modifiers for Errors  18
   - 1.3.2 Format Modifiers for the INPUT Statement  18
   - 1.3.3 Controlling Delimited Input  20
   - 1.3.4 Reading Variable-Length Records  24

1.4 Writing Delimited Files  28
   - 1.4.1 Using the DATA Step with the DLM= Option  28
   - 1.4.2 PROC EXPORT  29
   - 1.4.3 Using the %DS2CSV Macro  30
   - 1.4.4 Using ODS and the CSV Destination  31
   - 1.4.5 Inserting the Separator Manually  31

1.5 SQL Pass-Through  32
   - 1.5.1 Adding a Pass-Through to Your SQL Step  32
   - 1.5.2 Pass-Through Efficiencies  33

1.6 Reading and Writing to XML  33
   - 1.6.1 Using ODS  34
   - 1.6.2 Using the XML Engine  34

Chapter 2 Working with Your Data 37

2.1 Data Set Options 38
  2.1.1 REPLACE and REPEMPTY 40
  2.1.2 Password Protection 41
  2.1.3 KEEP, DROP, and RENAME Options 42
  2.1.4 Observation Control Using FIRSTOBS and OBS Data Set Options 43

2.2 Evaluating Expressions 45
  2.2.1 Operator Hierarchy 45
  2.2.2 Using the Colon as a Comparison Modifier 46
  2.2.3 Logical and Comparison Operators in Assignment Statements 47
  2.2.4 Compound Inequalities 49
  2.2.5 The MIN and MAX Operators 50
  2.2.6 Numeric Expressions and Boolean Transformations 51

2.3 Data Validation and Exception Reporting 52
  2.3.1 Date Validation 52
  2.3.2 Writing to an Error Data Set 55
  2.3.3 Controlling Exception Reporting with Macros 58

2.4 Normalizing - Transposing the Data 60
  2.4.1 Using PROC TRANSPOSE 61
  2.4.2 Transposing in the DATA Step 63

2.5 Filling Sparse Data 65
  2.5.1 Known Template of Rows 65
  2.5.2 Double Transpose 67
  2.5.3 Using COMPLETYPES with PROC MEANS or PROC SUMMARY 70
  2.5.4 Using CLASSDATA 70
  2.5.5 Using Preloaded Formats 72
  2.5.6 Using the SPARSE Option with PROC FREQ 73

2.6 Some General Concepts 73
  2.6.1 Shorthand Variable Naming 73
  2.6.2 Understanding the ORDER= Option 77
  2.6.3 Quotes within Quotes within Quotes 79
  2.6.4 Setting the Length of Numeric Variables 81

2.7 WHERE Specifics 82
  2.7.1 Operators Just for the WHERE 83
  2.7.2 Interaction with the BY Statement 86

2.8 Appending Data Sets 88
  2.8.1 Appending Data Sets Using the DATA Step and SQL UNION 88
  2.8.2 Using the DATASETS Procedure’s APPEND Statement 90
2.9  Finding and Eliminating Duplicates  90
  2.9.1  Using PROC SORT  91
  2.9.2  Using FIRST. and LAST. BY-Group Processing  92
  2.9.3  Using PROC SQL  93
  2.9.4  Using PROC FREQ  93
  2.9.5  Using the Data Component Hash Object  94

2.10  Working with Missing Values  97
  2.10.1  Special Missing Values  97
  2.10.2  MISSING System Option  98
  2.10.3  Using the CMISS, NMISS, and MISSING Functions  99
  2.10.4  Using the CALL MISSING Routine  100
  2.10.5  When Classification Variables are Missing  100
  2.10.6  Missing Values and Macro Variables  101
  2.10.7  Imputing Missing Values  101

Chapter  3  Just In the DATA Step  103

3.1  Working across Observations  105
  3.1.1  BY-Group Processing—Using FIRST. and LAST. Processing  105
  3.1.2  Transposing to ARRAYs  107
  3.1.3  Using the LAG Function  108
  3.1.4  Look-Ahead Using a MERGE Statement  110
  3.1.5  Look-Ahead Using a Double SET Statement  111
  3.1.6  Look-Back Using a Double SET Statement  111
  3.1.7  Building a FIFO Stack  113
  3.1.8  A Bit on the SUM Statement  114

3.2  Calculating a Person's Age  114
  3.2.1  Simple Formula  115
  3.2.2  Using Functions  116
  3.2.3  The Way Society Measures Age  117

3.3  Using DATA Step Component Objects  117
  3.3.1  Declaring (Instantiating) the Object  119
  3.3.2  Using Methods with an Object  119
  3.3.3  Simple Sort Using the HASH Object  120
  3.3.4  Stepping through a Hash Table  121
  3.3.5  Breaking Up a Data Set into Multiple Data Sets  126
  3.3.6  Hash Tables That Reference Hash Tables  128
  3.3.7  Using a Hash Table to Update a Master Data Set  130

3.4  Doing More with the INTNX and INTCK Functions  132
  3.4.1  Interval Multipliers  132
  3.4.2  Shift Operators  133
  3.4.3  Alignment Options  134
  3.4.4  Automatic Dates  136
3.5 Variable Conversions 138
   3.5.1 Using the PUT and INPUT Functions 138
   3.5.2 Decimal, Hexadecimal, and Binary Number Conversions 143

3.6 DATA Step Functions 143
   3.6.1 The ANY and NOT Families of Functions 144
   3.6.2 Comparison Functions 145
   3.6.3 Concatenation Functions 147
   3.6.4 Finding Maximum and Minimum Values 147
   3.6.5 Variable Information Functions 148
   3.6.6 New Alternatives and Functions That Do More 154
   3.6.7 Functions That Put the Squeeze on Values 163

3.7 Joins and Merges 165
   3.7.1 BY Variable Attribute Consistency 166
   3.7.2 Variables in Common That Are Not in the BY List 169
   3.7.3 Repeating BY Variables 170
   3.7.4 Merging without a Clear Key (Fuzzy Merge) 171

3.8 More on the SET Statement 172
   3.8.1 Using the NOBS= and POINT= Options 172
   3.8.2 Using the INDSNAME= Option 174
   3.8.3 A Comment on the END= Option 175
   3.8.4 DATA Steps with Two SET Statements 175

3.9 Doing More with DO Loops 176
   3.9.1 Using the DOW Loop 176
   3.9.2 Compound Loop Specifications 178
   3.9.3 Special Forms of Loop Specifications 178

3.10 More on Arrays 180
   3.10.1 Array Syntax 180
   3.10.2 Temporary Arrays 181
   3.10.3 Functions Used with Arrays 182
   3.10.4 Implicit Arrays 183

Chapter 4 Sorting the Data 185

4.1 PROC SORT Options 186
   4.1.1 The NODUPREC Option 186
   4.1.2 The DUPOUT= Option 187
   4.1.3 The TAGSORT Option 188
   4.1.4 Using the SORTSEQ Option 188
   4.1.5 The FORCE Option 190
   4.1.6 The EQUALS or NOEQUALS Options 190

4.2 Using Data Set Options with PROC SORT 190

4.3 Taking Advantage of Known or Knowable Sort Order 191
4.4 Metadata Sort Information 193
4.5 Using Threads 194

Chapter 5 Working with Data Sets 197
5.1 Automating the COMPARE Process 198
5.2 Reordering Variables on the PDV 200
5.3 Building and Maintaining Indexes 202
   5.3.1 Introduction to Indexing 203
   5.3.2 Creating Simple Indexes 204
   5.3.3 Creating Composite Indexes 206
   5.3.4 Using the IDXWHERE and IDXNAME Options 206
   5.3.5 Index Caveats and Considerations 207
5.4 Protecting Passwords 208
   5.4.1 Using PROC PWENCODE 208
   5.4.2 Protecting Database Passwords 209
5.5 Deleting Data Sets 211
5.6 Renaming Data Sets 211
   5.6.1 Using the RENAME Function 212
   5.6.2 Using PROC DATASETS 212

Chapter 6 Table Lookup Techniques 213
6.1 A Series of IF Statements—The Logical Lookup 215
6.2 IF-THEN/ELSE Lookup Statements 215
6.3 DATA Step Merges and SQL Joins 216
6.4 Merge Using Double SET Statements 218
6.5 Using Formats 219
6.6 Using Indexes 221
   6.6.1 Using the BY Statement 222
   6.6.2 Using the KEY= Option 222
6.7 Key Indexing (Direct Addressing)—Using Arrays to Form a Simple Hash 223
   6.7.1 Building a List of Unique Values 223
   6.7.2 Performing a Key Index Lookup 224
   6.7.3 Using a Non-Numeric Index 226
6.8 Using the HASH Object 227
Part 2  Data Summary, Analysis, and Reporting  231

Chapter  7  MEANS and SUMMARY Procedures  233

7.1 Using Multiple CLASS Statements and CLASS Statement Options  234
   7.1.1 MISSING and DESCENDING Options  236
   7.1.2 GROUPINTERNAL Option  237
   7.1.3 Order= Option  238

7.2 Letting SAS Name the Output Variables  238

7.3 Statistic Specification on the OUTPUT Statement  240

7.4 Identifying the Extremes  241
   7.4.1 Using the MAXID and MINID Options  241
   7.4.2 Using the IDGROUP Option  243
   7.4.3 Using Percentiles to Create Subsets  245

7.5 Understanding the _TYPE_ Variable  246

7.6 Using the CHARTYPE Option  248

7.7 Controlling Summary Subsets Using the WAYS Statement  249

7.8 Controlling Summary Subsets Using the TYPES Statement  250

7.9 Controlling Subsets Using the CLASSDATA= and EXCLUSIVE Options  251

7.10 Using the COMPLETETYPES Option  253

7.11 Identifying Summary Subsets Using the LEVELS and WAYS Options  254

7.12 CLASS Statement vs. BY Statement  255

Chapter  8  Other Reporting and Analysis Procedures  257

8.1 Expanding PROC TABULATE  258
   8.1.1 What You Need to Know to Get Started  258
   8.1.2 Calculating Percentages Using PROC TABULATE  262
   8.1.3 Using the STYLE= Option with PROC TABULATE  265
   8.1.4 Controlling Table Content with the CLASSDATA Option  267
   8.1.5 Ordering Classification Level Headings  269

8.2 Expanding PROC UNIVARIATE  270
   8.2.1 Generating Presentation-Quality Plots  270
   8.2.2 Using the CLASS Statement  273
   8.2.3 Probability and Quantile Plots  275
   8.2.4 Using the OUTPUT Statement to Calculate Percentages  276

8.3 Doing More with PROC FREQ  277
   8.3.1 OUTPUT Statement in PROC FREQ  277
   8.3.2 Using the NLEVELS Option  279
8.4 Using PROC REPORT to Better Advantage 280
8.4.1 PROC REPORT vs. PROC TABULATE 280
8.4.2 Naming Report Items (Variables) in the Compute Block 280
8.4.3 Understanding Compute Block Execution 281
8.4.4 Using a Dummy Column to Consolidate Compute Blocks 283
8.4.5 Consolidating Columns 284
8.4.6 Using the STYLE= Option with LINES 285
8.4.7 Setting Style Attributes with the CALL DEFINE Routine 287
8.4.8 Dates within Dates 288
8.4.9 Aligning Decimal Points 289
8.4.10 Conditionally Executing the LINE Statement 290

8.5 Using PROC PRINT 291
8.5.1 Using the ID and BY Statements Together 291
8.5.2 Using the STYLE= Option with PROC PRINT 292
8.5.3 Using PROC PRINT to Generate a Table of Contents 295

Chapter 9 SAS/GRAPH Elements You Should Know—Even if You Don’t Use SAS/GRAPH 297
9.1 Using Title Options with ODS 298
9.2 Setting and Clearing Graphics Options and Settings 300
9.3 Using SAS/GRAPH Statements with Procedures That Are Not SAS/GRAPH Procedures 303
9.3.1 Changing Plot Symbols with the SYMBOL Statement 303
9.3.2 Controlling Axes and Legends 306
9.4 Using ANNOTATE to Augment Graphs 309

Chapter 10 Presentation Graphics—More than Just SAS/GRAPH 313
10.1 Generating Box Plots 314
10.1.1 Using PROC BOXPLOT 314
10.1.2 Using PROC GPLOT and the SYMBOL Statement 315
10.1.3 Using PROC SHEWHART 316
10.2 SAS/GRAPH Specialty Techniques and Procedures 317
10.2.1 Building Your Own Graphics Font 317
10.2.2 Splitting a Text Line Using JUSTIFY= 319
10.2.3 Using Windows Fonts 319
10.2.4 Using PROC GKPI 320
10.3 PROC FREQ Graphics 323
Chapter 11 Output Delivery System  325
  11.1 Using the OUTPUT Destination  326
    11.1.1 Determining Object Names  326
    11.1.2 Creating a Data Set  327
    11.1.3 Using the MATCH_ALL Option  330
    11.1.4 Using the PERSIST= Option  330
    11.1.5 Using MATCH_ALL= with the PERSIST= Option  331
  11.2 Writing Reports to Excel  332
    11.2.1 EXCELXP Tagset Documentation and Options  333
    11.2.2 Generating Multisheet Workbooks  334
    11.2.3 Checking Out the Styles  335
  11.3 Inline Formatting Using Escape Character Sequences  337
    11.3.1 Page X of Y  338
    11.3.2 Superscripts, Subscripts, and a Dagger  340
    11.3.3 Changing Attributes  341
    11.3.4 Using Sequence Codes to Control Indentations, Spacing, and Line Breaks  342
    11.3.5 Issuing Raw RTF Specific Commands  344
  11.4 Creating Hyperlinks  345
    11.4.1 Using Style Overrides to Create Links  345
    11.4.2 Using the LINK= TITLE Statement Option  347
    11.4.3 Linking Graphics Elements  348
    11.4.4 Creating Internal Links  350
  11.5 Traffic Lighting  352
    11.5.1 User-Defined Format  352
    11.5.2 PROC TABULATE  353
    11.5.3 PROC REPORT  354
    11.5.4 Traffic Lighting with PROC PRINT  355
  11.6 The ODS LAYOUT Statement  356
  11.7 A Few Other Useful ODS Tidbits  358
    11.7.1 Using the ASIS Style Attribute  358
    11.7.2 ODS RESULTS Statement  358

Part 3  Techniques, Tools, and Interfaces  361

Chapter 12 Taking Advantage of Formats  363
  12.1 Using Preloaded Formats to Modify Report Contents  364
    12.1.1 Using Preloaded Formats with PROC REPORT  365
    12.1.2 Using Preloaded Formats with PROC TABULATE  367
    12.1.3 Using Preloaded Formats with the MEANS and SUMMARY Procedures  369
12.2 Doing More with Picture Formats  370
  12.2.1 Date Directives and the DATATYPE Option  371
  12.2.2 Working with Fractional Values  373
  12.2.3 Using the MULT and PREFIX Options  374
  12.2.4 Display Granularity Based on Value Ranges – Limiting
     Significant Digits  376
12.3 Multilabel (MLF) Formats  377
  12.3.1 A Simple MLF  377
  12.3.2 Calculating Rolling Averages  378
12.4 Controlling Order Using the NOTSORTED Option  381
12.5 Extending the Use of Format Translations  382
  12.5.1 Filtering Missing Values  382
  12.5.2 Mapping Overlapping Ranges  383
  12.5.3 Handling Text within Numeric Values  383
  12.5.4 Using Perl Regular Expressions within Format Definitions  384
  12.5.5 Passing Values to a Function as a Format Label  384
12.6 ANYDATE Informats  388
  12.6.1 Reading in Mixed Dates  389
  12.6.2 Converting Mixed DATETIME Values  389
12.7 Building Formats from Data Sets  390
12.8 Using the PVALUE Format  392
12.9 Format Libraries  393
  12.9.1 Saving Formats Permanently  393
  12.9.2 Searching for Formats  394
  12.9.3 Concatenating Format Catalogs and Libraries  394

Chapter 13 Interfacing with the Macro Language  397
13.1 Avoiding Macro Variable Collisions—Make Your Macro Variables
    %Local  398
13.2 Using the SYMPUTX Routine  400
  13.2.1 Compared to CALL SYMPUT  401
  13.2.2 Using SYMPUTX to Save Values of Options  402
  13.2.3 Using SYMPUTX to Build a List of Macro Variables  402
13.3 Generalized Programs—Variations on a Theme  403
  13.3.1 Steps to the Generalization of a Program  403
  13.3.2 Levels of Generalization and Levels of Macro Language
     Understanding  405
13.4 Utilizing Macro Libraries  406
  13.4.1 Establishing an Autocall Library  406
  13.4.2 Tracing Autocall Macro Locations  408
  13.4.3 Using Stored Compiled Macro Libraries  408
  13.4.4 Macro Library Search Order  409
13.5  Metadata-Driven Programs  409
   13.5.1 Processing across Data Sets  409
   13.5.2 Controlling Data Validations  410
13.6  Hard-Coding—Just Don’t Do It  415
13.7  Writing Macro Functions  417
13.8  Macro Information Sources  420
   13.8.1 Using SASHelp and Dictionary tables  420
   13.8.2 Retrieving System Options and Settings  422
   13.8.3 Accessing the Metadata of a SAS Data Set  424
13.9  Macro Security and Protection  426
   13.9.1 Hiding Macro Code  426
   13.9.2 Executing a Specific Macro Version  427
13.10 Using the Macro Language IN Operator  430
   13.10.1 What Can Go Wrong  430
   13.10.2 Using the MINOPERATOR Option  431
   13.10.3 Using the MINDELIMITER= Option  432
   13.10.4 Compilation vs. Execution for these Options  432
13.11 Making Use of the MFILE System Option  433
13.12 A Bit on Macro Quoting  434

Chapter 14  Operating System Interface and Environmental Control  437
14.1  System Options  438
   14.1.1 Initialization Options  438
   14.1.2 Data Processing Options  441
   14.1.3 Saving SAS System Options  444
14.2  Using an AUTOEXEC Program  446
14.3  Using the Configuration File  446
   14.3.1 Changing the SASAUTOS Location  447
   14.3.2 Controlling DM Initialization  449
14.4  In the Display Manager  449
   14.4.1 Showing Column Names in ViewTable  450
   14.4.2 Using the DM Statement  451
   14.4.3 Enhanced Editor Options and Shortcuts  452
   14.4.4 Macro Abbreviations for the Enhanced Editor  456
   14.4.5 Adding Tools to the Application Tool Bar  461
   14.4.6 Adding Tools to Pull-Down and Pop-up Menus  463
   14.4.7 Adding Tools to the KEYS List  466
14.5  Using SAS to Write and Send E-mails  467
14.6 Recovering Physical Location Information 468
14.6.1 Using the PATHNAME Function 468
14.6.2 SASHELP VIEWS and DICTIONARY Tables 468
14.6.3 Determining the Executing Program Name and Path 469
14.6.4 Retrieving the UNC (Universal Naming Convention) Path 470

Chapter 15 Miscellaneous Topics 473
15.1 A Few Miscellaneous Tips 474
15.1.1 Customizing Your NOTEs, WARNINGs, and ERRORs 474
15.1.2 Enhancing Titles and Footnotes with the #BYVAL and #BYVAR Options 475
15.1.3 Executing OS Commands 477
15.2 Creating User-defined Functions Using PROC FCMP 479
15.2.1 Building Your Own Functions 479
15.2.2 Storing and Accessing Your Functions 481
15.2.3 Interaction with the Macro Language 482
15.2.4 Viewing Function Definitions 483
15.2.5 Removing Functions 484
15.3 Reading RTF as Data 485
15.3.1 RTF Diagram Completion 486
15.3.2 Template Preparation 486
15.3.3 RTF as Data 487

Appendix A Topical Index 489

Appendix B Usage Index 491
Global Statements and Options 492
Statements, Global 492
Macro Language 493
GOPTIONS, Graphics 493
Options, System 493
Options, Data Set 495

Procedures: Steps, Statements, and Options 495
Procedures 495

DATA Step: Statements and Options 500
Statements, DATA Step 500
Format Modifiers 501
Functions 501
Hash Object 504
Output Delivery System, ODS 504
  ODS Destinations and Tagsets 504
  ODS Attributes 505
  ODS Options 505
  ODS Statements 506

SAS Display Manager 506
  Display Manager Commands 506

References 507

User Publications 507

Generally Good Reading—Lots More to Learn 518
  SAS Documentation 518
  SAS Usage Notes 518
  Discussion Forums 518
  Newsletters, Corporate and Private Sites 519
  User Communities 519
  Publications 519
  Learning SAS 520

Index 521

Chapter 1
Moving, Copying, Importing, and Exporting Data

1.1 LIBNAME Statement Engines 4
   1.1.1 Using Data Access Engines to Read and Write Data 5
   1.1.2 Using the Engine to View the Data 6
   1.1.3 Options Associated with the Engine 6
   1.1.4 Replacing EXCEL Sheets 7
   1.1.5 Recovering the Names of EXCEL Sheets 8

1.2 PROC IMPORT and EXPORT 9
   1.2.1 Using the Wizard to Build Sample Code 9
   1.2.2 Control through the Use of Options 9
   1.2.3 PROC IMPORT Data Source Statements 10
   1.2.4 Importing and Exporting CSV Files 12
   1.2.5 Preventing the Export of Blank Sheets 15
   1.2.6 Working with Named Ranges 16

1.3 DATA Step INPUT Statement 17
   1.3.1 Format Modifiers for Errors 18
   1.3.2 Format Modifiers for the INPUT Statement 18
   1.3.3 Controlling Delimited Input 20
   1.3.4 Reading Variable-Length Records 24

1.4 Writing Delimited Files 28
   1.4.1 Using the DATA Step with the DLM= Option 28
   1.4.2 PROC EXPORT 29
   1.4.3 Using the %DS2CSV Macro 30
   1.4.4 Using ODS and the CSV Destination 31
   1.4.5 Inserting the Separator Manually 31
1.5 SQL Pass-Through 32
   1.5.1 Adding a Pass-Through to Your SQL Step 32
   1.5.2 Pass-Through Efficiencies 33

1.6 Reading and Writing to XML 33
   1.6.1 Using ODS 34
   1.6.2 Using the XML Engine 34

A great deal of the process of the preparation of the data is focused on the movement of data from one table to another. This transfer of data may be entirely within the control of SAS or it may be between disparate data storage systems. Although most of the emphasis in this book is on the use of SAS, not all data are either originally stored in SAS or even ultimately presented in SAS. This chapter discusses some of the aspects associated with moving data between tables as well as into and out of SAS.

When moving data into and out of SAS, Base SAS allows you only limited access to other database storage forms. The ability to directly access additional databases can be obtained by licensing one or more of the various SAS/ACCESS products. These products give you the ability to utilize the SAS/ACCESS engines described in Section 1.1 as well as an expanded list of databases that can be used with the IMPORT and EXPORT procedures (Section 1.2).

SEE ALSO
Andrews (2006) and Frey (2004) both present details of a variety of techniques that can be used to move data to and from EXCEL.

1.1 LIBNAME Statement Engines

In SAS® a number of engines are available for the LIBNAME statement. These engines allow you to read and write data to and from sources other than SAS. These engines can reduce the need to use the IMPORT and EXPORT procedures.

The number of available engines depends on which products your company has licensed from SAS. One of the most popular is SAS/ACCESS® Interface to PC Files.

You can quickly determine which engines are available to you. An easy way to build this list is through the NEW LIBRARY window.

From the SAS Explorer right click on LIBRARIES and select NEW. Available engines appear in the ENGINE pull-down list.

Pulling down the engine list box on the ‘New Library’ dialog box shown to the right, indicates the engines,
including the EXCEL engine, among others, which are available to this user.

PROC SETINIT can also be used to determine which products have been licensed.

The examples in this section show various aspects of the EXCEL engine; however, most of what is demonstrated can be applied to other engines as well.

**SEE ALSO**


### 1.1.1 Using Data Access Engines to Read and Write Data

In the following example, the EXCEL engine is used to create an EXCEL workbook, store a SAS data set as a sheet in that workbook, and then read the data back from the workbook into SAS.

```sas
libname toxls excel "&path\data\newwb.xls";
proc sort data=advrpt.demog
  out=toxls.demog;
  by clinnum;
run;

data getdemog;
  set toxls.demog;
  run;
libname toxls clear;
```

The use of the EXCEL engine establishes the TOXLS libref so that it can be used to convert to and from the Microsoft Excel workbook NEWWB.XLS. If it does not already exist, the workbook will be created upon execution of the LIBNAME statement.

For many of the examples in this book, the macro variable &PATH is assumed to have been defined. It contains the upper portion of the path appropriate for the installation of the examples on your system. See the book’s introduction and the AUTOEXEC.SAS in the root directory of the example code, which you may download from support.sas.com/authors.

Data sets that are written to the TOXLS libref will be added to the workbook as named sheets. This OUT= option adds a sheet with the name of DEMOG to the NEWWB.XLS workbook.

A sheet can be read from the workbook, and brought into the SAS world, simply by naming the sheet.

As should be the case with any libref, when you no longer need the association, the libref should be cleared. This can be especially important when using data engines, since as long as the libref exists, access to the data by applications other than SAS is blocked. Until the libref is cleared, we are not able to view or work with any sheets in the workbook using Excel.

**MORE INFORMATION**

LIBNAME statement engines are also discussed in Sections 1.1.2 and 1.2.6. The XML engine is discussed in Section 1.6.2.
1.1.2 Using the Engine to View the Data

Once an access engine has been established by a libref, we are able to do almost all of the things that we typically do with SAS data sets that are held in a SAS library.

The SAS Explorer shows the contents of the workbook with each sheet appearing as a data table.

When viewing an EXCEL workbook through a SAS/ACCESS engine, each sheet appears as a data set. Indeed you can use the VIEWTABLE or View Columns tools against what are actually sheets. Notice in this image of the SAS Explorer, that the DEMOG sheet shows up twice. Sheet names followed by a $ are actually named ranges, which under EXCEL can actually be a portion of the entire sheet. Any given sheet can have more than one named range, so this becomes another way to filter or subset what information from a given sheet will be brought into SAS through the SAS/ACCESS engine.

1.1.3 Options Associated with the Engine

The SAS/ACCESS engine is acting like a translator between two methods of storing information, and sometimes we need to be able to control the interface. This can often be accomplished through the use of options that modify the translation process. Many of these same options appear in the PROC IMPORT/EXPORT steps as statements or options.

It is important to remember that not all databases store information in the same relationship as does SAS. SAS, for instance, is column based - an entire column (variable) will be either numeric or character. EXCEL, on the other hand, is cell based – a given cell can be considered numeric, while the cell above it in the same column stores text. When translating from EXCEL to SAS we can use options to establish guidelines for the resolution of ambiguous situations such as this.

Connection Options

For database systems that require user identification and passwords these can be supplied as options on the LIBNAME statement.

- **USER** User identification
- **PASSWORD** User password
- **others** Other connection options vary according to the database to which you are connecting

LIBNAME Statement Options

These options control how information that is passed through the interface is to be processed. Most of these options are database specific and are documented in the sections dealing with your database.
When working with EXCEL typical LIBNAME options might include:

- **HEADER** Determines if a header row exists or should be added to the table.
- **MIXED** Some columns contain both numeric and character information.
- **VER** Controls which type (version) of EXCEL is to be written.

### Data Source Options

Some of the same options associated with PROC IMPORT (see Section 1.2.3) can also be used on the LIBNAME statement. These include:

- **GETNAMES** Incoming variable names are available in the *first row* of the incoming data.
- **SCANTEXT** A length is assigned to a character variable by scanning the incoming column and determining the maximum length.

### 1.1.4 Replacing EXCEL Sheets

While the EXCEL engine allows you to establish, view, and use a sheet in an Excel workbook as a SAS data set, you cannot update, delete or replace the sheet from within SAS. It is possible to replace the contents of a sheet, however, with the help of PROC DATASETS and the SCAN_TEXT=NO option on the LIBNAME statement. The following example shows how to replace the contents of an EXCEL sheet.

In the first DATA step the programmer has ‘accidently’ used a WHERE clause that writes the incorrect data, in this case 0 observations, to the EXCEL sheet. Simply correcting and rerunning the DATA step will not work because the sheet already exists.

We could step out of SAS and use EXCEL to manually remove the bad sheet; however, we would rather do it from within SAS. First we must reestablish the libref using the SCAN_TEXT=NO option. PROC DATASETS can then be used to *delete* the sheet. In actuality the sheet has not truly been deleted, but merely cleared of all contents. Since the sheet is now truly empty and the SCAN_TEXT option is set to NO, we can now replace the empty sheet with the desired contents.
The DATA step can now be rerun ②, and the sheet contents will now be correct. When SAS has completed its work with the workbook, and before you can use the workbook using EXCEL you will need to clear the libref. This can be done using the CLEAR option on the LIBNAME statement ③.

MORE INFORMATION
See Section 1.2 for more information on options and statements in PROC IMPORT and PROC EXPORT. In addition to PROC DATASETS, Section 5.4 discusses other techniques that can be used to delete tables. Section 14.4.5 also has an example of deleting data sets using PROC DATASETS.

SEE ALSO
Choate and Martell (2006) discuss this and numerous other techniques that can be used with EXCEL.

### 1.1.5 Recovering the Names of EXCEL Sheets

Especially when writing automated systems you may need to determine the names of workbook sheets. There are a couple of ways to do this.

If you know the libref(s) of interest, the automatic view SASHELP.VTABLE can be used in a DATA step to see the sheet names. This view contains one observation for every SAS data set in every SAS library in current use, and for the TOXLS libref the sheet names will be shown as data set names.

```sas
data toxls.ClinicNames; ②
  set advrpt.clinicnames;
run;
libname toxls clear; ⑤
```

When there are a number of active libraries, the process of building this table can be lengthy. As a general rule using the DICTIONARY.MEMBERS table in a PROC SQL step has a couple of advantages. It is usually quicker than the SASHELP.VTABLE view, and it also has an ENGINE column which allows you to search without knowing the specific libref.

```sas
data sheetnames;
  set sashelp.vtable;
  where libname = 'TOXLS';
run;
```

```sas
proc sql;
  create table sheetnames as
  select * from dictionary.members
  where engine= 'EXCEL' ;
quit ;
```

The KEEP statement or the preferred KEEP= data set option could have been used in these examples to reduce the number of variables (see Section 2.1.3).

MORE INFORMATION
SASHELP views and DICTIONARY tables are discussed further in Section 13.8.1.

SEE ALSO
A thread in the SAS Forums includes similar examples.
http://communities.sas.com/thread/10348?tstart=0
1.2 PROC IMPORT and EXPORT

Like the SAS/ACCESS engines discussed in Section 1.1, the IMPORT and EXPORT procedures are used to translate data into and out of SAS from a variety of data sources. The SAS/ACCESS product, which is usually licensed separately through SAS (but may be bundled with Base SAS), controls which databases you will be able to move data to and from. Even without SAS/ACCESS you can still use these two procedures to read and write text files such as comma separated variables (CSV), as well as files using the TAB and other delimiters to separate the variables.

1.2.1 Using the Wizard to Build Sample Code

The import/export wizard gives you a step-by-step guide to the process of importing or exporting data. The wizard is easy enough to use, but like all wizards does not lend itself to automated or batch processing. Fortunately the wizard is actually building a PROC IMPORT/EXPORT step in the background, and you can capture the completed code. For both the import and export process the last screen prompts you to ‘Create SAS Statements.’

```
PROC EXPORT DATA= WORK.A
  OUTFILE= "C:\temp\junk.xls" DBMS=EXCEL
  REPLACE;
RUN;
```

The following PROC EXPORT step was built using the EXPORT wizard. A simple inspection of the code indicates what needs to be changed for a future application of the EXPORT procedure. Usually this means that the wizard itself needs to be run infrequently.

1. The DATA= option identifies the data set that is to be converted.
2. In this case, since we are writing to EXCEL the OUTFILE= identifies the workbook.
3. If the sheet already exists, it will be replaced.
4. The sheet name can also be provided.

Converting the previous generic step to one that creates a CSV file is very straightforward.

```
PROC EXPORT DATA= sashelp.class
  OUTFILE= "C:\path\data\class.csv" DBMS=csv
  REPLACE;
RUN;
```

SEE ALSO

Raithel (2009) discusses the use of the EXPORT wizard to generate code in a sasCommunity.org tip.

1.2.2 Control through the Use of Options

There are only a few options that need to be specified. Of these most of the interesting ones are used when the data are being imported (clearly SAS already knows all about the data when it is being exported).
▪ **DBMS=** Identifies the incoming database structure (including .CSV and .TXT). Since database structures change with versions of the software, you should know the database version. Specific engines exist at the version level for some databases (especially Microsoft’s EXCEL and ACCESS). The documentation discusses which engine is optimized for each software version.

▪ **REPLACE** Determines whether or not the destination target (data set, sheet, table) is replaced if it already exists.

### 1.2.3 PROC IMPORT Data Source Statements

These statements give you additional control over how the incoming data are to be read and interpreted. Availability of any given source statement depends on the type (DBMS=) of the incoming data.

▪ **DATAROW** First incoming row that contains data.

▪ **GETNAMES** The names of the incoming columns are available in the first row of the incoming data. Default column names when none are available on the incoming table are VAR1, VAR2, etc.

▪ **GUESSINGROWS** Number of rows SAS will scan before determining if an incoming column is numeric or character. This is especially important for mixed columns and early rows are all numeric. In earlier versions of SAS modifications to the SAS Registry were needed to change the number of rows used to determine the variable’s type, which is fortunately no longer necessary.

▪ **RANGE and SHEET** For spreadsheets a specific sheet name, named range, or range within a sheet can be specified.

▪ **SCANTEXT and TEXTSIZE** PROC IMPORT assigns a length to a character variable by scanning the incoming column and determining the maximum.

When using GETNAMES to read column names from the source data, keep in mind that most databases use different naming conventions than SAS and may have column names that will cause problems when imported. By default illegal characters are replaced with an underscore (_) by PROC IMPORT. When you need the original column name, the system option VALIDVARNAME=ANY (see Section 14.1.2) allows a broader range of acceptable column names.
In the contrived data for the following example we have an EXCEL file containing a subject number and a response variable (SCALE). The import wizard can be used to generate a PROC IMPORT step that will read the XLS file (MAKESCALE.XLS) and create the data set WORK.SCALEDATA. This PROC IMPORT step creates two numeric variables.

Notice that the form of the supporting statements is different than form most procedures. They look more like options (option=value;) than like statements. The GETNAMES= statement is used to determine the variable names from the first column.

When importing data SAS must determine if a given column is to be numeric or character. A number of clues are utilized to make this determination. SAS will scan a number of rows for each column to try to determine if all the values are numeric. If a non-numeric value is found, the column will be read as a character variable; however, only some of the rows are scanned and consequently an incorrect determination is possible. The MIXED= statement is used to specify that the values in a given column are always of a single type (numeric or character). When set to YES, the IMPORT procedure will tend to create character variables in order to accommodate mixed types.

In this contrived example it turns out that starting with subject 271 the variable SCALE starts taking on non-numeric values. Using the previous PROC IMPORT step does not detect this change, and creates SCALE as a numeric variable. This, of course, means that data will be lost as SCALE will be missing for the observations starting from row 712.

For PROC IMPORT to correctly read the information in SCALE it needs to be a character variable. We can encourage IMPORT to create a character variable by using the MIXED and GUESSINGROWS statements.
Changing the MIXED= value to YES ➊ is not necessarily sufficient to cause SCALE to be a character value; however, if the value of the DBMS option is changed from EXCEL to XLS ➋, the MIXED=YES statement ➌ is honored and SCALE is written as a character variable in the data set SCALEDATA.

When MIXED=YES is not practical the GUESSINGROWS= statement can sometimes be used to successfully determine the type for a variable.

GUESSINGROWS cannot be used when DBMS=EXCEL, however it can be used when DBMS=XLS. Since GUESSINGROWS ➍ changes the number of rows that are scanned prior to determining if the column should be numeric or character, its use can increase the time and resources required to read the data.

SEE ALSO

### 1.2.4 Importing and Exporting CSV Files

Comma Separated Variable, CSV, files have been a standard file type for moving data between systems for many years. Fortunately we now have a number of superior tools available to us so that we do not need to resort to CSV files as often. Still they are commonly used and we need to understand how to work with them.

Both the IMPORT and EXPORT procedures can work with CSV files (this capability is a part of the Base SAS product and a SAS/ACCESS product is not required). Both do the conversion by first building a DATA step, which is then executed.

#### Building a DATA Step

When you use the import/export wizard to save the PROC step (see Section 1.2.1), the resulting DATA step is not saved. Fortunately you can still get to the generated DATA step by recalling the last submitted code.

1. Execute the IMPORT/EXPORT procedure.
2. While in the Display Manager, go to RUN→Recall Last Submit.

Once the code generated by the procedure is loaded into the editor, you can modify it for other purposes or simply learn from it. For the simple PROC EXPORT step in Section 1.2.1, the following code is generated:

```sas
PROC IMPORT OUT= WORK.scaledata
    DATAFILE= "C:\Temp\makescale.xls"
    DBMS=xls REPLACE;
    GETNAMES=YES; ➍
    GUESSINGROWS=800; ➎
RUN;
```
### Headers are Not on Row 1

The ability to create column names based on information contained in the data is very beneficial. This is especially important when building a large SAS table from a CSV file with lots of columns. Unfortunately we do not always have a CSV file with the column headers in row 1. Since GETNAMES=YES assumes that the headers are in row 1 we cannot use GETNAMES=YES. Fortunately this is SAS, so there are alternatives.

The CSV file created in the PROC EXPORT step in Section 1.2.1 has been modified so that the column names are on row 3. The first few lines of the file are:  

```sas
%**********************************************************************
* PRODUCT:   SAS
* VERSION:   9.1
* CREATOR:   External File Interface
* DATE:      11APR09
* DESC:      Generated SAS Datastep Code
* TEMPLATE SOURCE:  (None Specified.)
%**********************************************************************

data _null_;            end=EFIEOD;
set SASHELP.CLASS;      %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
%let _EFIREC_ = 0;       /* clear export record count macro variable */
file 'C:\InnovativeTechniques\data\class.csv' delimiter=',';
DSD DROPOVER lrec=32767;
format Name $8. ;
format Sex $1. ;
format Age best12. ;
format Height best12. ;
format Weight best12. ;
if _n_ = 1 then        /* write column names */
do;
    put 'Name'      /* write column names */
       ',      /* write column names */
       'Sex'      /* write column names */
       ',      /* write column names */
       'Age'      /* write column names */
       ',      /* write column names */
       'Height'    /* write column names */
       ',      /* write column names */
       'Weight'    /* write column names */
       ;
end;
do;
EFIOUT + 1;
put Name $ @;
put Sex $ @;
put Age @;
put Height @;
put Weight ;
end;
if _ERROR_ then call symputx('_EFIERR_','1'); /*set ERROR detection macro variable*/
run;
```
Carpenter’s Guide to Innovative SAS Techniques

The DATA step generated by PROC IMPORT (E1_2_3c_ImportWO.SAS), simplified somewhat for this example, looks something like:

```
data WORK.CLASSWO ;
  infile "&path\Data\classwo.csv" delimiter = ',';
    MISSOVER DSD lrecl=32767 firstobs=4 ;
    informat VAR1 $8. ;
    informat VAR2 $1. ;
    informat VAR3 best32. ;
    informat VAR4 best32. ;
    informat VAR5 best32. ;
    format VAR1 $8. ;
    format VAR2 $1. ;
    format VAR3 best12. ;
    format VAR4 best12. ;
    format VAR5 best12. ;
  input
    VAR1 $ ;
    VAR2 $ ;
    VAR3 ;
    VAR4 ;
    VAR5 ;
  ;
run;
```

Class Data from SASHELP,,,
Comma Separated rows; starting in row 3,,,
Name,Sex,Age,Height,Weight
Alfred,M,14,69,112.5
Alice,F,13,56.5,84
Barbara,F,13,65.3,98
Carol,F,14,62.8,102.5

... data not shown ... 

Clearly SAS has substituted VAR1, VAR2, and so on for the unknown variable names. If we knew the variable names, all we would have to do to fix the problem would be to rename the variables.

The following macro reads the header row from the appropriate row in the CSV file, and uses that information to rename the columns in WORK.CLASSWO.
SEE ALSO
McGuown (2005) also discusses the code generated by PROC IMPORT when reading a CSV file. King (2011) uses arrays and hash tables to read CSV files with unknown or varying variable lists. These flexible and efficient techniques could be adapted to the type of problem described in this section.

1.2.5 Preventing the Export of Blank Sheets

PROC EXPORT does not protect us from writing a blank sheet when our exclusion criteria excludes all possible rows from a given sheet. In the following example we have inadvertently asked to list all students with SEX='q'. There are none of course, and the resulting sheet is blank, except for the column headers.
We can prevent this from occurring by first identifying those levels of SEX that have one or more rows. There are a number of ways to generate a list of values of a variable; however, an SQL step is ideally suited to place those values into a macro variable for further processing.

The name of the data set that is to be exported, as well as the classification variable, are passed to the macro %MAKEXLS as named parameters.

```
%macro makexls(dsn=,class=);
%local valuelist listnum i value;
proc sql noprint;
select distinct &class into :valuelist separated by ' ' from &dsn;
%let listnum = &sqlobs;
quit;

/* One export for each sheet; */
%do i = 1 %to &listnum;
  %let value = %scan(&valuelist,&i,%str( ));
  proc export data=&dsn(where=((&class="&value"))
    outfile="c:\temp\&dsn..xls"
    dbms=excel2000
    replace;
    SHEET="&class:&value";
  run;
%end;
%mend makexls;
%makexls(dsn=sashelp.class,class=sex)
```

SEE ALSO
A similar example which breaks a data set into separate sheets can be found in the article “Automatically_Separating_Data_into_Excel_Sheets” on sasCommunity.org.
http://www.sascommunity.org/wiki/Automatically_Separating_Data_into_Excel_Sheets

1.2.6 Working with Named Ranges
By default PROC IMPORT and the LIBNAME statement’s EXCEL engine expect EXCEL data to be arranged in a certain way (column headers, if present, on row one column A; and data starting on row two). It is not unusual, however, for the data to be delivered as part of a report or as a subset of a larger table. One solution is to manually cut and paste the data onto a blank sheet so that it conforms to the default layout. It can often be much easier to create a named range.
Chapter 1: Moving, Copying, Importing, and Exporting Data

1.3 DATA Step INPUT Statement

The INPUT statement is loaded with options that make it extremely flexible. Since there has been a great deal written about the basic INPUT statement, only a few of the options that seem to be under used have been collected here.

SEE ALSO
An overview about reading raw data with the INPUT statement can be found in the SAS documentation at http://support.sas.com/publishing/pubcat/chaps/58369.pdf. Schreier (2001) gives a short overview of the automatic _INFILE_ variable along with other information regarding the reading of raw data.
1.3.1 Format Modifiers for Errors

Inappropriate data within an input field can cause input errors that prevent the completion of the data set. As the data are read, a great many messages can also be generated and written to the LOG. The (?) and (??) format modifiers control error handling. Both the ? and the ?? suppress error messages in the LOG; however, the ?? also resets the automatic error variable (_ERROR_) to 0. This means that while both of these operators control what is written to the LOG only the ?? will necessarily prevent the step from terminating when the maximum error count is reached.

In the following step, the third data row contains an invalid value for AGE. AGE is assigned a missing value, and because of the ?? operator no ‘invalid data’ message is written to the LOG.

```
data base;
input age ?? name $;
datalines;
15   Fred
14   Sally
x    John
run;
```

MORE INFORMATION

The ?? modifier is used with the INPUT function in Sections 2.3.1 and 3.6.1.

SEE ALSO

The SAS Forum thread found at http://communities.sas.com/message/48729 has an example that uses the ?? format modifier.

1.3.2 Format Modifiers for the INPUT Statement

Some of the most difficult input coding occurs when combining the use of informats with LIST style input. This style is generally required when columns are not equally spaced so informats can’t be easily used, and the fields are delimited with blanks. LIST is also the least flexible input style. Informat modifiers include:

- & allows embedded blanks in character variables
- : allows the use of informats for non-aligned columns
- ~ allows the use of quotation marks within data fields

Because of the inherent disadvantages of LIST input (space delimited fields), when it is possible, consider requesting a specific unique delimiter. Most recently generated files of this type utilize a non-blank delimiter, which allows you to take advantage of some of the options discussed in Section 1.3.3. Unfortunately many legacy files are space delimited, and we generally do not have the luxury of either requesting a specific delimiter or editing the existing file to replace the spaces with delimiters.

There are two problems in the data being read in the following code. The three potential INPUT statements (two of the three are commented) highlight how the ampersand and colon can be used to help read the data. Notice that DOB does not start in a consistent column and the second last name has an embedded blank.
Using the first INPUT statement without informat modifiers \(1\) shows, that for the second data line, both the date and the last name have been read incorrectly.

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Sam</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Susan</td>
<td>Mc Callister</td>
<td>10/10/2019</td>
</tr>
</tbody>
</table>

Assuming the second INPUT statement \(2\) was commented and used, the colon modifier is placed in front of the date informat. The colon allows the format to essentially float to the appropriate starting point by using LIST input and then applying the informat once the value is found.

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Sam</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Mc Callister</td>
<td>Susan</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>

The birthdays are now being read correctly; however, Susan’s last name is being split because the embedded blank is being interpreted as a field delimiter. The ampersand \(3\) can be used to allow embedded spaces within a field.

By placing an ampersand after the variable name (LNAME) \(4\), the blank space becomes part of the variable rather than a delimiter. We are now reading both the date of birth and the last name correctly.

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Sam</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Mc Callister</td>
<td>Susan</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>

While the ampersand is also used as a macro language trigger, this will not be a problem when it is used as an INPUT statement modifier as long as it is not immediately followed by text that could be interpreted as a macro variable name (letter or underscore). In this example the ampersand is followed by the semicolon so there will be no confusion with the macro language.
While the trailing ampersand can be helpful it can also introduce problems as well. If the data had been slightly more complex, even this solution might not have worked. The following data also contains a city name. Even though the city is not being read, the trailing & used with the last name (LNAME) causes the city name to be confused with the last name.

Because of the trailing & and the length of LNAME ($15) a portion of the city (New York) has been read into the LNAME for the second observation. On the first observation the last name is correct because more than one space separates Johnson and Seattle. Even with the trailing &, more than one space is still successfully seen as a field delimiter.

On the second observation the city would not have been confused with the last name had there been two or more spaces between the two fields.

1. Placing the FORMAT statement within the DATA step causes the format to be associated with the variable DOB in subsequent steps. The INFORMAT statement is only used when reading the data.

2. The DATALINES statement causes subsequent records to be read as data up to, but not including, the first line that contains a semicolon. In the previous examples the RUN statement doubles as the end of data marker. Many programmers use a separate semicolon to perform this task. Both styles are generally considered acceptable (as long as you are using the RUN statement to end your step).

With only a single space between the last name and the city, the trailing & alone is not sufficient to help the INPUT statement distinguish between these two fields. Additional variations of this example can be found in Section 1.3.3.

MORE INFORMATION

LIST input is a form of delimited input and as such these options also apply to the examples discussed in Section 1.3.3. When the date form is not consistent one of the *any date* informats may be helpful. See Section 12.6 for more information on the use of these specialized informats.

SEE ALSO


### 1.3.3 Controlling Delimited Input

Technically LIST input is a form of delimited input, with the default delimiter being a space. This means that the modifiers shown in Section 1.3.2 apply to other forms of delimited input, including comma separated variable, CSV, files.
**INFILE Statement Options**
Options on the INFILE statement are used to control how the delimiters are to be interpreted.

- **DELIMITER** Specifies the character that delimits fields (other than the default - a space). This option is often abbreviated as DLM=.
- **DLMSTR** Specifies a single multiple character string as a delimiter.
- **DLMOPT** Specifies parsing options for the DLMSTR option.
- **DSD** Allows character fields that are surrounded by quotes (by setting the comma as the delimiter). Two successive delimiters are interpreted as individual delimiters, which allow missing values to be assigned appropriately. DSD also removes quotation marks from character values surrounded by quotes. If the comma is not the delimiter you will need to use the DLM= option along with the DSD option.

Some applications, such as Excel, build delimiter separated variable files with quotes surrounding the fields. This can be critical if a field’s value can contain the field separator. For default list input, where a space is a delimiter, it can be very difficult to successfully read a field with an embedded blank (see Section 1.3.2 which discusses the use of trailing & to read embedded spaces). The DSD option alerts SAS to the potential of quoted character fields. The following example demonstrates simple comma-separated data.

```plaintext
data base;
  length lname $15;
  infile datalines dlm=',';
  *infile datalines dlm=',' dsd;
  input fname $ lname $ dob :mmddyy10.;
  datalines;
  'Sam','Johnson',12/15/1945
  'Susan','Mc Callister',10/10/1983
run;
```

1.3.3a Delimited List Input Modifiers

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Johnson'</td>
<td>'Sam'</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>'Mc Callister'</td>
<td>'Susan'</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>

The fields containing character data have been quoted. Since we do not actually want the quote marks to be a part of the data fields, the DSD option alerts the parser to this possibility and the quotes themselves become a part of the field delimiting process.

```plaintext
infile datalines dlm=',', dsd;
```

Using the DSD option results in data fields without the quotes.

```plaintext
1.3.3a Delimited List Input Modifiers

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Sam</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Mc Callister</td>
<td>Susan</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>
```

---

1. Although the INFILE statement is often not needed when using the DATALINES, CARDS, or CARDS4 statements, it can be very useful when the options associated with the INFILE statement are needed. The *fileref* can be DATALINES or CARDS.

The DLM= option is used to specify the delimiter. In this example the field delimiter is specified as a comma.

2. Using the DSD option results in data fields without the quotes.
On the INPUT Statement
The tilde (~) can be used to modify a format, much the same way as a colon (:); however, the two modifiers are not exactly the same.

```
title '1.3.3b Delimited List Input Modifiers';
title2 'Using the ~ Format Modifier';
data base;
length lname $15;
infile datalines dlm=', ' dsd;
input fname $ lname $ birthloc $~15. dob :mmddyy10. ;
datalines;
'Sam','Johnson', 'Fresno, CA', '12/15/1945'
'Susan','Mc Callister', 'Seattle, WA', '10/10/1983'
r
```

The tilde format modifier correctly reads the BIRTHLOC field; however, it preserves the quote marks that surround the field. Like the colon, the tilde can either precede or follow the $ for character variables. As an aside notice that for this example quote marks surround the numeric date value for the first row. The field is still processed correctly as a numeric SAS date value.

```
1.3.3b Delimited List Input Modifiers
Using the ~ Format Modifier

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>fname</th>
<th>birthloc</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Sam</td>
<td>'Fresno, CA'</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Mc Callister</td>
<td>Susan</td>
<td>'Seattle, WA'</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>
```

Replacing the tilde (~) with a colon (:) would cause the BIRTHLOC value to be saved without the quote marks. If instead we supply a length for BIRTHLOC ($), neither a format nor the tilde will be needed.

```
title '1.3.3c Delimited List Input Modifiers';
title2 'BIRTHLOC without a Format Modifier';
title3 'BIRTHLOC Length Specified';
data base;
length lname $15;
infile datalines dlm=', ' dsd;
in input fname $ lname $ birthloc $ dob :mmddyy10. ;
datalines;
'Sam','Johnson', 'Fresno, CA','12/15/1945'
'Susan','Mc Callister', 'Seattle, WA', '10/10/1983'
run;
```

```
1.3.3c Delimited List Input Modifiers
BIRTHLOC without a Format Modifier
BIRTHLOC Length Specified

<table>
<thead>
<tr>
<th>Obs</th>
<th>lname</th>
<th>birthloc</th>
<th>fname</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnson</td>
<td>Fresno, CA</td>
<td>Sam</td>
<td>12/15/1945</td>
</tr>
<tr>
<td>2</td>
<td>Mc Callister</td>
<td>Seattle, WA</td>
<td>Susan</td>
<td>10/10/1983</td>
</tr>
</tbody>
</table>
```
Multiple Delimiters

It is possible to read delimited input streams that contain more than one delimiter. In the following small example two delimiters, a comma and a slash are both used to delimit the data values.

```
data imports;
  infile cards dlm='/,';
  input id importcode $ value;
  cards;
    14,1,13
    25/Q9,15
    6,D/20
run;
```

```
Obs   id    importcode    value
1     14        1 13
2     25        Q9 15
3      6        D 20
```

Notice that the DLM option causes either the comma or the slash to be used as field delimiters, but not the slash comma together as a single delimiter (see the DLMSTR option below to create a single multiple character delimiter).

Because the INFILE statement is executed for each observation, the value assigned to the DLM option does not necessarily need to be a constant. It can also be a variable or can be changed using IF-THEN/ELSE logic. In the simplest form this variable could be assigned in a retain statement.

This simple example demonstrates a delimiter that varies by observation. Here the first character of each line is the delimiter that is to be used in that line. The delimiter is read, stored, and then used on the INFILE statement. Here we are taking advantage of the executable nature of the INFILE statement.

Using DLMSTR

Unlike the DLM option, which designates one or more delimiters, the DLMSTR option declares a specific list of characters to use as a delimiter. Here the delimiter is the sequence of characters comma-comma-slash (,,/). Notice in the LISTING of the IMPORT data set, that extra commas and slashes are read as data.

```
data imports;
  infile cards;
  input dlmvar $1. @;
  infile cards dlm=dlmvar;
  input @2 id importcode $ value;
  cards;
    ,14,1,13
    /25/Q9/15
    ~6~D~20
run;
```

```
1.3.3g Use a delimiter string
Obs   id    importcode    value
1     14       1/ 13
2     25       Q9, 15
3      6       ,D 20
```
SEE ALSO
The following SAS Forum thread discussed the use of the DLM and DLMSTR options http://communities.sas.com/message/46192. The use of the tilde when writing data was discussed on the following forum thread: http://communities.sas.com/message/57848. The INFILE and FILE statements are discussed in more detail by First (2008).

1.3.4 Reading Variable-Length Records
For most raw data files, including the small ones shown in most of the preceding examples, the number of characters on each row has not been consistent. Inconsistent record length can cause problems with lost data and incomplete fields. This is especially true when using the formatted style of input. Fortunately there are several approaches to reading this kind of data successfully.

The Problem Is
Consider the following data file containing a list of patients. Unless it has been built and defined as a fixed-length file, which is very unlikely on most operating systems including Windows, each record has a different length. The individual records physically stop after the last non-blank character. When we try to read the last name on the third row (Rachel’s last name is unknown), we will be attempting to read past the end of the physical record and there will almost certainly be an error.

The following code attempts to read the above data. However, we have a couple of problems.

```sas
filename patlist "&path\data\patientlist.txt";
data patients;
infile patlist;
input @2  sex $1.
     @8  fname $10.
     @18 lname $15.;
run;
title '1.3.4a Varying Length Records';
proc print data=patients;
run;
```

The LOG shows two notes; there is a LOST CARD and the INPUT statement reached past the end of the line.
The resulting data set has a number of data problems. Even a quick inspection of the data shows that the data fields have become confused.

```
NOTE: LOST CARD.
sex=M fname=Adam lname= _ERROR_=1 _N_=6
NOTE: 12 records were read from the infile PATLIST.
The minimum record length was 13.
The maximum record length was 26.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
```

The TRUNCOVER option is specified and as much information as possible is gathered from each record; however, SAS does not go to the next physical record to complete the observation.

### 1.3.4a Varying Length Records

<table>
<thead>
<tr>
<th>Obs</th>
<th>sex</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>Linda</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>M</td>
<td>Mat</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>Terrie</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>Merv</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>Robert</td>
<td>M</td>
</tr>
</tbody>
</table>

**INFILE Statement Options (TRUNCOVER, MISSOVER)**

Two INFILE statement options can be especially useful in controlling how SAS handles short records.

- **MISSOVER**
  - Assigns missing values to variables beyond the end of the physical record. Partial variables are set to missing.

- **TRUNCOVER**
  - Assigns missing values to variables beyond the end of the physical record. Partial variables are truncated, but not necessarily set to missing.

- **FLOWOVER**
  - SAS finishes the logical record using the next physical record. This is the default.

```
title '1.3.4b Varying Length Records';
title2 'Using TRUNCOVER';
data patients(keep=sex fname lname);
infile patlist truncover;
input @2  sex $1.
     @8  fname $10.
     @18 lname $15.;
run;
```
Generally the TRUNCOVER option is easier to apply than the $VARYING informat, and there is no penalty for including a TRUNCOVER option on the INFILE statement even when you think that you will not need it.

By including the TRUNCOVER option on the INFILE statement, we have now correctly read the data without skipping a record, while correctly assigning a missing value to Rachel’s last name.

Using the $VARYING Informat

The $VARYING informat was created to be used with variable-length records. This informat allows us to determine the record length and then use that length for calculating how many columns to read. As a general rule, you should first attempt to use the more flexible and easier to apply TRUNCOVER option on the INFILE statement, before attempting to use the $VARYING informat.

Unlike other informats $VARYING utilizes a secondary value to determine how many bytes to read. Very often this value depends on the overall length of the record. The record length can be retrieved with the LENGTH= option and a portion of the overall record length is used to read the field with a varying width.

The classic use of the $VARYING informat is shown in the following example, where the last field on the record has an inconsistent width from record to record. This is also the type of data read for which the TRUNCOVER option was designed.

The LENGTH= option on the INFILE statement specifies a temporary variable (LEN) which holds the length of the current record. An INPUT statement with just a trailing @ is used to load the record into the input buffer. Here the length is determined and loaded into the variable LEN. The trailing @ holds the record so that it can be read again.

The width of the last name is calculated (total length less the number of characters to the left of the name). The variable NAMEWIDTH holds this value for use by the $VARYING informat.

---

### 1.3.4b Varying Length Records
**Using TRUNCOVER**

<table>
<thead>
<tr>
<th>Obs</th>
<th>sex</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>Linda</td>
<td>Maxwell</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>Ronald</td>
<td>Mercy</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>Rachel</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>Mat</td>
<td>Most</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>David</td>
<td>Nabers</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>Terrie</td>
<td>Nolan</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>June</td>
<td>Olsen</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>Merv</td>
<td>Panda</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>Mathew</td>
<td>Perez</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>Robert</td>
<td>Pope</td>
</tr>
<tr>
<td>11</td>
<td>M</td>
<td>Arthur</td>
<td>Reilly</td>
</tr>
<tr>
<td>12</td>
<td>M</td>
<td>Adam</td>
<td>Robertson</td>
</tr>
</tbody>
</table>

### 1.3.4c Varying Length Records
**Using the $VARYING Informat**

<table>
<thead>
<tr>
<th>Obs</th>
<th>sex</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>Linda</td>
<td>Maxwell</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>Ronald</td>
<td>Mercy</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>M</td>
<td>Mat</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>David</td>
<td>Nabers</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>Terrie</td>
<td>Nolan</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>June</td>
<td>Olsen</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>Merv</td>
<td>Panda</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>Mathew</td>
<td>Perez</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>Robert</td>
<td>Pope</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>Arthur</td>
<td>Reilly</td>
</tr>
<tr>
<td>11</td>
<td>M</td>
<td>Adam</td>
<td>Robertson</td>
</tr>
</tbody>
</table>

---

Using the $VARYING Informat

<table>
<thead>
<tr>
<th>Obs</th>
<th>sex</th>
<th>fname</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>Linda</td>
<td>Maxwell</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>Ronald</td>
<td>Mercy</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>M</td>
<td>Mat</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>David</td>
<td>Nabers</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>Terrie</td>
<td>Nolan</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>June</td>
<td>Olsen</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>Merv</td>
<td>Panda</td>
</tr>
<tr>
<td>8</td>
<td>M</td>
<td>Mathew</td>
<td>Perez</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>Robert</td>
<td>Pope</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>Arthur</td>
<td>Reilly</td>
</tr>
<tr>
<td>11</td>
<td>M</td>
<td>Adam</td>
<td>Robertson</td>
</tr>
</tbody>
</table>

---

The LENGTH= option on the INFILE statement specifies a temporary variable (LEN) which holds the length of the current record.

The width of the last name is calculated (total length less the number of characters to the left of the name). The variable NAMEWIDTH holds this value for use by the $VARYING informat.
The width of the last name field for this particular record follows the $VARYING15. informat. Here the width used with the $VARYING informat is the widest possible value for LNAME and also establishes the variable’s length.

Inspection of the resulting data shows that we are now reading the correct last name; however, we still have a data issue for the third and fourth input lines. Since the third data line has no last name, the $VARYING informat jumps to the next data record. The TRUNCOVER option on the INFILE statement discussed above addresses this issue successfully.

In fact for the third record the variable FNAME, which uses a $10 informat, reaches beyond the end of the record and causes the data to be misread.

Using a LENGTH statement to declare the variable lengths avoids the need to add a width to the informats.

Neither a first or last name is included. This code assumes that a gender (SEX) is always present.

The record is too short to have a last name, but must contain a first name of at least one letter.

The last name must have at least one letter.

NAMEWIDTH will contain the width of the rightmost variable. The value of this variable is generally of no interest, but it is kept here so that you can see its values change for each observation.

It is easy to see that the $VARYING informat is more difficult to use than either the TRUNCOVER or the MISSOVER options. However, the $VARYING informat can still be helpful. In the following simplified example suggested by John King there is no delimiter and yet the columns are not of constant width. To make things more interesting the variable with the inconsistent width is not on the end of the input string.

The first field (WIDTH) contains the number of characters in the second field (DATANAME). This value is used with the $VARYING NAME informat to correctly read the data set name while not reading past the name and into the next field (DATACODE).
SEE ALSO
Cates (2001) discusses the differences between MISSOVER and TRUNCOVER. A good comparison of these options can also be found in the SAS documentation http://support.sas.com/documentation/cdl/en/basess/58133/HTML/default/viewer.htm#a002645812.htm.

SAS Technical Support example #37763 uses the $VARYING. informat to write a zero-length string in a REPORT example http://support.sas.com/kb/37/763.html.

1.4 Writing Delimited Files

Most modern database systems utilize metadata to make the data itself more useful. When transferring data to and from Excel, for instance, SAS can take advantage of this metadata. Flat files do not have the advantage of metadata and consequently more information must be transferred through the program itself. For this reason delimited data files should not be our first choice for transferring information from one database system to another. That said we do not always have that choice. We saw in Section 1.3 a number of techniques for reading delimited data.

Since SAS already knows all about a given SAS data set (it has access to the metadata), it is much more straightforward to write delimited files.

MORE INFORMATION
Much of the discussion on reading delimited data also applies when writing delimited data (see Section 1.3).

1.4.1 Using the DATA Step with the DLM= Option

When reading delimited data using the DATA step, the INFILE statement is used to specify a number of controlling options. Writing the delimited file is similar; however, the FILE statement is used. Many of the same options that appear on the INFILE statement can also be used on the FILE statement. These include:

- DLM=
- DLMSTR=
- DSD

While the DSD option by default implies a comma as the delimiter, there are differences between the uses of these two options. The DSD option will cause values which contain an embedded delimiter character to be double quoted. The DSD option also causes missing values to appear as two consecutive delimiters, while the DLM= alone writes the missing as either a period or a blank.
In the following example three columns from the ADVRPT.DEMOG data set are to be written to the comma separated variable (CSV) file. The FILE statement is used to specify the delimiter using the DLM= option. Just in case one of the fields contains the delimiter (a comma in this example), the Delimiter Sensitive Data option, DSD ④, is also included. Using the DSD option is a good general practice.

When you also want the first row to contain the column names, a conditional PUT statement can be used to write them. The data itself is also written using a PUT statement ⑤.

### MORE INFORMATION
The example in Section 1.4.4 shows how to insert the header row without explicitly naming the variables.

All the variables on the PDV can be written by using the statement PUT (_ALL_) ( ); (see Section 1.4.5).

### 1.4.2 PROC EXPORT

Although a bit less flexible than the DATA step, the EXPORT procedure is probably easier to use for simple cases. However, it has some characteristics that make it ‘not so easy’ when the data are slightly less straightforward.

The EXPORT step shown here is intended to mimic the output file generated by the DATA step in Section 1.4.1; however, it is not successful and we need to understand why.

```sas
filename outspot "&path\data\E1_4_2demog.csv";
proc export data=advrpt.demog(keep=fname lname dob) ⑥
    outfile=outspot ②
dbms=csv ③ replace;
delimiter=''; ④
run;
```

① Three variables have been selected from ADVRPT.DEMOG and EXPORT is used to create a CSV file.

② The OUTFILE= option points to the fileref associated with the file to be created. Notice that the extension of the file’s name matches the selected database type ③.

③ The DBMS= option is used to declare the type for the generated file. In this case a CSV file. Other choices include TAB and DLM (and others if one of the SAS/ACCESS products has been licensed).

④ The DELIMITER= option is used to designate the delimiter. It is not necessary in this example as the default delimiter for a CSV file is a comma. This option is most commonly used when DBMS is set to DLM and something other than a space, the default delimiter for DBMS=DLM, is desired as the delimiter.
A quick inspection of the file generated by the PROC EXPORT step shows that all the variables from the ADVRPT.DEMOG data set have been included in the file; however, only those variables in the KEEP= data set option have values. Data set options cannot be used with the incoming data set when EXPORT creates delimited data. Either you will need to write all the variables or the appropriate variables need to be selected in a previous step (see Section 1.4.3). This behavior is an artifact of the way that PROC EXPORT writes the delimited file. PROC EXPORT writes a DATA step and builds the variable list from the metadata, ignoring the data set options. When the data are actually read into the constructed DATA step; however, the KEEP= data set option is applied, thus resulting in the missing values.

```
subject,clnnum,lname,fname,ssn,sex,dobdeath,race,edu,wt,ht,symp,death2
,,Adams,Mary,,,12AUG51,,,,,,,,
,,Adamson,Joan,,,,,,,,
,,Alexander,Mark,,,15JAN30,,,,,,,,
,,Antler,Peter,,,15JAN34,,,,,,,,
,,Atwood,Teddy,,,14FEB50,,,,,,,,
....
```

1.4.3 Using the %DS2CSV Macro

The DS2CSV.SAS file is a macro that ships with Base SAS, and is accessed through the SAS autocall facility. Its original authorship predates many of the current capabilities discussed elsewhere in Section 1.4. The macro call is fairly straightforward; however, the macro code itself utilizes SCL functions and lists and is outside the scope of this book.

The macro is controlled through the use of a series of named or keyword parameters. Only a small subset of this list of parameters is shown here.

```
data part;
    set advrpt.demog(keep=fname lname dob);
run;
%ds2csv(data=part,
    runmode=b,
    labels=n,
    csvfile=&path\data\E1_4_3demog.csv)
```

1. The data set to be processed is passed to the macro.
2. The data set to be processed is passed to the macro.
3. By default the variable labels are used in the column header. Generally you will want the column names to be passed to the CSV file. This is done using the LABELS= parameter.
4. The CSVFILE= parameter is used to name the CSV file. This parameter does not accept a fileref.

SEE ALSO

A search of SAS documentation for the macro name, DS2CSV, will surface the documentation for this macro.
1.4.4 Using ODS and the CSV Destination

The Output Delivery System, ODS, and the CSV tagset can be used to generate CSV files. When you want to create a CSV file of the data, complete with column headers, the CSV destination can be used in conjunction with PROC PRINT.

```sas
ods csv file="&path\data\E1_4_4demog.csv"
   options(doc='Help'
   delimiter='",');
proc print data=advrpt.demog
   noobs;
   var fname lname dob;
run;
ods csv close;
```

1. The new delimited file is specified using the FILE= option.
2. TAGSET options are specified in the OPTIONS list. A list of available options can be seen using the DOC='HELP' option.
3. The delimiter can be changed from a comma with the DELIMITER= option.
4. The OBS column is removed using the NOOBS option.
5. Select variables and variable order using the VAR statement in the PROC PRINT step.
6. As always be sure to close the destination.

MORE INFORMATION
Chapter 11 discusses a number of aspects of the Output Delivery System.

SEE ALSO
There have been several SAS forum postings on the CSV destination.
http://communities.sas.com/message/29026
http://communities.sas.com/message/19459

1.4.5 Inserting the Separator Manually

When using the DATA step to create the delimited file, the techniques shown in Section 1.4.1 will generally be sufficient. However you may occasionally require more control, or you may want to take control of the delimiter more directly.

One suggestion that has been seen in the literature uses the PUT statement to insert the delimiter.

```sas
data _null_;
   set advrpt.demog(keep=fname lname dob);
   file csv_a;
   if _n_=1 then put 'FName,LName,DOB';
   put (_all_)(',');
run;
```

Here the _ALL_ variable list shortcut has been used to specify that all variables are to be written. This shortcut list requires a corresponding text, format, or other modifier for each of the variables. In this case we have specified a comma, e.g., (',') 1.

This approach will work to some extent, but it is not perfect in that a comma precedes each line of data.
The DSD option on the FILE statement implies a comma as the delimiter, although the DLM= option can be used to specify a different option (see Section 1.4.1). The _ALL_ list abbreviation can still be used; however, a neutral modifier must also be selected. Either the colon (:) or the question mark (?) will serve the purpose.

Because the DSD option has been used, an approach such as this one will also work when one or more of the variables contain an embedded delimiter.

### 1.5 SQL Pass-Through

SQL pass through allows the user to literally pass instructions through a SAS SQL step to the server of another database. Passing code or SQL instructions out of the SQL step to the server can have a number of advantages, most notably significant efficiency gains.

#### 1.5.1 Adding a Pass-Through to Your SQL Step

The pass-through requires three elements to be successful:

- A connection must be formed to the server/database.
- Code must be passed to the server/database.
- The connection must be closed.

These three elements will be formulated as statements (CONNECT and DISCONNECT) or as a clause within the FROM CONNECTION phrase.

```
proc sql noprint;
   connect to odbc (dsn=clindat uid=Susie pwd=pigtails);
   create table stuff as select * from connection to odbc (select * from q.org for fetch only);
   disconnect from odbc;
quit;
```

The connection that is established using the CONNECT statement and is then referred to in the FROM CONNECTION TO phrase.

Notice that the SQL code that is being passed to the database, not a SAS database, is within the parentheses. This code must be appropriate for the receiving database. In this case the pass through is to a DB2 table via an ODBC connection.

There are a number of types of connections and while ODBC connections, such as the one established in this example, are almost universally available in the Microsoft/Windows world, they are typically slower than SAS/ACCESS connections.
1.5.2 Pass-Through Efficiencies

When using PROC SQL to create and pass database-specific code to a database other than SAS, such as Oracle or DB2, it is important that you be careful with how you program the particular problem. Depending on how it is coded SQL can be very efficient or very inefficient, and this can be an even more important issue when you use pass-through techniques to create a data subset.

Passing information back from the server is usually slower than processing on the server. Design the pass-through to minimize the amount of returned information. Generally the primary database will be stored at a location with the maximum processing power. Take advantage of that power. At the very least minimizing the amount of information that has to be transferred back to you will help preserve your bandwidth.

In SQL, data sets are processed in memory. This means that large data set joins should be performed where available memory is maximized. When a join becomes memory bound subsetting the data before the join can be helpful. Know and understand your database and OS, some WHERE statements form clauses that are applied to the result of the join rather than to the incoming data set.

Even when you do not intend to write to the primary database that is being accessed using an SQL pass-through, extra process checking may be involved against that data table. These checks, which can be costly, can potentially be eliminated by designating the incoming data table as read-only. This can be accomplished in a number of ways. In DB2 using the clause for fetch only in the code that is being passed to the database eliminates write checks against the incoming table. In the DB2 pass-through example in Section 1.5.1 we only want to extract or fetch data. We speed up the process by letting the database know that we will not be writing any data – only fetching it.

MORE INFORMATION
An SQL step using pass-through code can be found in Section 5.4.2.

1.6 Reading and Writing to XML

Extensible Markup Language, XML, has a hierarchical structure while SAS data sets are record or observation based. Because XML is fast becoming a universal data exchange format, it is incumbent for the SAS programmer to have a working knowledge of how to move information from SAS to XML and from XML to SAS.

The XML engine (Section 1.6.2) was first introduced in Version 8 of SAS. Later the ODS XML destination was added; however, currently the functionality of the XML destination has been built into the ODS MARKUP destination (see Section 1.6.1).

Because XML is text based and each row contains its own metadata, the files themselves can be quite large.

SEE ALSO
A very nice overview of XML and its relationship to SAS can be found in (Pratter, 2008). Other introductory discussions on the relationship of XML to SAS include: Chapal (2003), Palmer (2003 and 2004), and in the SAS documentation on “XML Engine with DATA Step or PROC COPY”.
1.6.1 Using ODS

You can create an XML file using the ODS MARKUP destination. The file can contain procedure output in XML form, and this XML file can then be passed to another application that utilizes / reads XML. By default the MARKUP destination creates a XML file.

1. The FILE= option is used to designate the name of the file to be created. Notice the use of the XML extension.

2. The procedure must be within the ODS `sandwich`.

3. The destination must be closed before the file 1 can be used outside of SAS.

MORE INFORMATION

If the application that you are planning to use with the XML file is Excel, the EXCELXP tagset is a superior choice (see Section 11.2).

SEE ALSO

The LinkedIn thread http://www.linkedin.com/groupItem?view=&srchtype=discussedNews&gid=70702&item=74453221&type=member&trk=b_it-dig-b_pd-ttl-cn&ut=34c4-P0gjofkY1 follows a discussion of the generation of XML using ODS.

1.6.2 Using the XML Engine

The use of the XML engine is a process similar to the one shown in Section 1.6.1, and can be used to write to the XML format. XML is a markup language and XML code is stored in a text file that can be both read and written by SAS. As in the example above, an engine is used on the LIBNAME statement to establish the link with SAS that performs the conversion. A fileref is established and it is used in the LIBNAME statement.

1. On the LIBNAME statement that has the XML engine, the XMLFILEREF= option is used to point to the fileref either containing the XML file or, as is the case in this example, the file that is to be written.
The `libref` `TOXML` can be used to both read and write the XML file. The name of the data set (PATLIST) is recorded as a part of the XML file. This means that multiple SAS data sets can be written to the same XML file.

The selected variables are written to the XML file. Notice that the variables are named on each line and that the date has been re-coded into a YYYY-MM-DD form, and that the missing DOB for 'Joan Adamson' has been written using the `missing=` notation.

**SEE ALSO**
Hemedinger and Slaughter (2011) briefly describe the use of XML and the XML Mapper.

Index

A

absolute column references 281
ACROSS option
   DEFINE statement (REPORT) 281–282, 284
   LEGEND statement 308
ACTUAL= option, HEBULLETS statement (GKPI) 321
Add Abbreviation dialog box 456
Add Action dialog box 464
ADD method 120, 123
age calculations
   about 114–115
   functions for 116–117, 419
   simple formula for 115–116
   society measuring age 117
%AGE macro function 419
AGE statement, DATASETS procedure 212
aliases, report items and 281
aligning
decimal points 289–290
texting across rows 341
ALL keyword 89, 261–262, 278
   _ALL_ list abbreviation
      DATASETS procedure and 76
      inserting separators manually 31–32
      SORT procedure and 187
ALTER data set option 41
   -ALTLOG initialization option 439–440
ampersand (&) 19–20, 434–435
ANALYSIS option, DEFINE statement (REPORT) 281
ANCHOR= option, ODS PDF statement 349–351
anchor tags (HTML) 295
AND operator 85
ANGLE= option, AXIS statement 307
ANNO= option 273, 309–311
annotate facility 273, 309–311
ANNOTATE= option 309
ANYALNUM function 144
ANYALPHA function 144–145, 161
ANYDATE informats 388–390
ANYDIGIT function 144
ANYDSTDTE. informat 388–390
ANYDSTDTE10. informat 389
ANYDSTDTE. informat 388–390
ANYDSTDTE. informat 388
ANYSPACE function 144
ANYUPPER function 144
ANYXDIGIT function 143–144
APPEND option, CONFIG.CFG file 447
APPEND statement, DATASETS procedure 90
Appender object 118
appending data sets 88–90
application toolbar, adding tools to 461–462
ARCCOS function 154
ARRAY statement
   key indexing and 224
   reordering variables on PDV and 202
   shorthand variable naming and 73–74
   syntax for 180–181
   temporary arrays and 181
   transposing data example 64
arrays
   about 180
   functions used within 182–183
   implicit 183–184
   key indexing and 223–227
   shorthand variable naming and 73–74
   syntax for 180–181
   table lookup techniques 214
   temporary 181
   transposing data to 64, 107–108
ASCENDING option
   CLASS statement (MEANS) 234
   CLASS statement (SUMMARY) 234
ASCII collating sequence 188
ASIS style attribute 358
ASSIGN Keys dialog box 454, 460
assignment statements, logical and comparison operators in 47–49
asterisk (*) 202, 410
at sign (@) 26, 340
AUTOCALL macro libraries 406–408
-AUTOEXEC initialization option 439, 448
-AUTOEXEC.SAS program 446
AUTOLABEL option, OUTPUT statement 239–240
automatic dates 136–138
automatic variables
   See specific automatic variables
automating processes 198–200, 329
AUTONAME option, OUTPUT statement 239–240
AutoSave feature (Enhanced Editor) 455
AVG. format 379
AXIS statement
   about 306
   ANGLE= option 307
   COLOR= option 307
   FONT= option 307
   generating box plots 315, 317
   HEIGHT= option 307
   LABEL= option 307
   MAJOR= option 307
   MINOR= option 307
   ORDER= option 307
   ROTATE= option 308
   UNIVARIATE procedure and 273
   VALUE= option 307
B
%B directive 372
%b directive 372
BACKGROUND= attribute 266
BCOLOR= option
   FOOTNOTE statement 298–299
   TITLE statement 298–299
BEEP command 453
BEST. format 139
BEST32. format 169
BETWEEN operator 83
BINARY. format 143
binary number conversions 143
BMI (Body Mass Index) 310–311, 321, 481
BMP files 439
Body Mass Index (BMI) 310–311, 321, 481
BODYTITLE option, ODS RTF statement 299, 338–339
BOLD option
    FOOTNOTE statement  298
    TITLE statement  298
Boolean transformations  51–52
BORDER graphics option  300
BOX= option, TABLE statement (TABULATE)  261, 265
box plots, generating  314–317
BOXPLOT procedure
    about  314–315
    high-resolution graphs and  303
    PLOT statement  314–315
BOXSTYLE option, PLOT statement (BOXPLOT)  314
BOXWIDTH option, PLOT statement (BOXPLOT)  314
BOXWIDTHSCALE option, PLOT statement (BOXPLOT)  314
%BQUOTE macro function  435
_BREAK_ automatic variable  281
BWIDTH= option, SYMBOL statement  316
BY-group processing
    eliminating duplicate observations  92–93
    FIRST. processing and  92–93, 105–107, 123
    indexes and  203
    LAST. processing and  92–93, 105–107
    WHERE statement and  86–88
BY statement
    CLASS statement and  255
    ID statement and  291–292
    indexes and  222
    MERGENOBY= system option and  441
    percentile statistics example  245
    PRINT procedure  291–292
    SORT procedure  121
    table lookup techniques  216, 222
    TRANPOSE procedure  199
    UNIVARIATE procedure  328
BY variables
    attribute consistency  166–169
    common to data sets  169–170
    FREQ procedure and  475
    repeating  170–171
    UNIVARIATE procedure and  328
    #BYLINE option, TITLE statement  476
    #BYVAL option
        FOOTNOTE statement  475–476
        TITLE statement  245, 338–339, 475–476
    #BYVR option
        FOOTNOTE statement  475–476
        TITLE statement  245, 338–339, 475–476
C
    calculations
        moving averages  107, 113–114, 378–380
        person’s age  114–117, 419
    CALL DEFINE routine
        REPORT procedure and  79
        style attributes and  287–288
        style overrides and  345–346
        traffic lighting and  354–356
    CALL EXECUTE routine  414–415, 483
    CALL MISSING routine
        about  100, 148
        arrays and  183
        building FIFO stacks  113
        eliminating duplicate observations  96
        transposing data to arrays  108
    CALL MODULE routine  470–472
    CALL PRNTCRT routine  483
    CALL SYMPUT routine  401–402
    CALL SYMPUTX routine
        about  400
        building list of macro variables  402–403
    CALL SYMPUT routine and  401–402
    %GETGLOBAL macro and  440
    IF statement processing and  163, 179–180
    saving values of options  402
    CALL SYSTEM routine  478
    CAPABILITY procedure  303, 317
    CARDS statement  21
    CARD$ statement  21
    Cartesian product  171
    case-sensitive reordering  189
    CASE statement, SQL procedure  215
    CASE_FIRST keyword  189
    CAT function  147
    CATALOG procedure  211, 395
    catalogs
        concatenating  394–395
        deleting  211
        renaming  212
        saving formats  393
        saving informats  393
    CATQ function  147
    CATS function  147, 163, 403
    CATT function  147, 295
    CATX function  147, 163
    CEIL function  46
    CELLWIDTH= attribute  287
    C2F function  481
    C2FF function  386–387
    CHANGE statement, DATASETS procedure  212
    _CHARACTER_ variable name list  76, 99
    character variables
        CMISS function and  99–100
        shorthand naming  75–76
        variable conversions and  138–142
    CHARTYPE option
        MEANS procedure  247–248
        SUMMARY procedure  247–248
    CHECK method  130
    CHISQ option, TABLE statement (FREQ)  278, 323
    CLASS statement, GLM procedure  100
    CLASS statement, MEANS procedure
        ASCENDING option  234
        BY statement and  255
        DESCENDING option  234
        EXCLUSIVE option  235, 369–370
        generalizing programs example  404
        GROUPINTERNAL option  235, 237
        missing classification variables and  100
        MISSING option  100, 234–236
        MLF option  235
        ORDER= option  78, 235, 237–238
        ordered data and  191–192
        PRELOADFMT option  235, 369
        sort considerations  191–193
    CLASS statement, SUMMARY procedure
        ASCENDING option  234
        BY statement and  255
        DESCENDING option  234, 236
        EXCLUSIVE option  235, 369–370
        GROUPINTERNAL option  235, 237
        MISSING option  100, 234–236
        MLF option  235
        ORDER= option  78, 192, 235, 237–238
        ordered data and  191–192
        PRELOADFMT option  235, 369
CSV files
  additional information 15
  importing/exporting 12–15
  writing 29–32
CTEXT= graphics option 302
CTITLE= graphics option 302
CTONUM. informat 141
Customize Tools dialog box 461–462
%D directive 373
dagger symbol 340–341
dash (-) 438–441
data engines
  additional information 5
  clearing librefs and 5
determining availability of 4
  LIBNAME statement and 4–8
  options associated with 6–7
  reading and writing data with 5
  replacing Excel sheets with 7–8
viewing data 6
data normalization
  about 60–61
  TRANSPOSE procedure and 61–63
  transposing in DATA steps 63–64
DATA= option
  COMPARE procedure 198–199
  DELETE procedure 211
  EXPORT procedure 9
  TRANSPOSE procedure 61
data processing options 441–444
data set options
  about 38–39
  controlling observations 42–45
currenting replacement conditions 40–41
DATA step statements and 41–42, 206–207
  ODS OUTPUT statement and 328
  password protection 41
  SORT procedure and 190–191
data sets
  accessing metadata for 424–426
  appending 88–90
  automating processes and 198–200, 329
  breaking up 126–128
  building and maintaining indexes 202–207
  building formats from 390–392
  creating 327–329
  deleting 211
  indexes and 207
  processing metadata across 409–410
  protecting passwords 208–210
  recovering physical location information 468–472
  renaming 211–212
  reordering variables on PDV 200–202
  updating with hash tables 130–131
data source statements 10–12
Data Step Component Interface
  See DSCI (Data Step Component Interface)
DATA steps
  See also specific DO loops
  See also specific statements and functions
  accessing metadata of data sets 424–426
  alternative functions 154–163
  ANY family of functions 144–145
  appending data sets 88–90
  arrays in 180–184
  building 12–14
  calculating person’s age 114–117, 419
  comparison functions 145–147
  component objects in 117–131
  concatenation functions 147
  counting functions 155
  creating indexes 203–205, 221
  data set options 39, 206–207
determining unique keys 94–95
  eliminating duplicate observations 95–96
  executing OS commands 478
  finding minimum/maximum values 50–51, 147–148
generating e-mails 467
  HASH objects and 227–229
  IN comparison operator and 47, 430
  joins and merges in 165–171, 216–218
  NOT family of functions 144–145
  powerful and flexible functions 154–163
  processing across observations 105–114
  transposing data in 63–64
  underutilized functions 143–165
  variable conversions 138–143
  variable information functions 148–154
  WHERE usage in 82–83
data validation
  about 52
  checking date strings 53–54
  in metadata-driven programs 410–415
database passwords 209–210
DATALINES statement 20–21
DATA NULL step 120–121, 126–127
DATAROW statement, IMPORT procedure 10
DATASET: constructor 95, 119, 121
DATASETS procedure
  AGE statement 212
  APPEND statement 90
  ATTRIB statement 76
  CHANGE statement 212
  copying index files 207
  creating indexes 203–205, 221
  DELETE statement 211
  deleting data sets 211
  deleting sheets 7
  INDEX statement 205, 222
  KILL option 211
  MEMTYPE= option 211
  MODIFY statement 76, 222
  NOLIST option 211, 222
  DATASTMTCHK system option 40–41, 442
%DATATYP macro function 145
DATATYPE= option, PICTURE statement (FORMAT) 371–373
%DATAVAL macro 414
date directives 371–373
DATE function 385
date manipulation
  intervals and ranges 137
  nested dates 288–289
date values 371–373, 385–386
%DATEC. format 386
DATAE. format 386
DATEPART function 385
dates
  automatic 136–138
  building date-specific formats 371–373
  checking strings with formats 53–55
  collapsing 136–137
  expanding 137
  intervals/ranges for 137
  previous month by name 137–138
  reading in mixed dates 389
DATESTYLE system option 389
DATETIME function 385
datetime values 371–373, 385–386, 389–390
DAY function 117
DBMS= option
EXPORT procedure 10, 29
IMPORT procedure 10, 12
debugging macro programs 210, 403–405, 433
decimal number conversions 143
decimal points, aligning 289–290
DECLARE statement
about 119–120
eliminating duplicate observations 95
HASH objects and 228
hash tables referencing hash tables 129
simple sort example 121
DEFAULT= option, VALUE statement (FORMAT) 384
DEFINE routine
See CALL DEFINE routine
DEFINE statement, REPORT procedure
ACROSS option 281–282, 284
ANALYSIS option 281
COMPUTED option 281
DISPLAY option 281
GROUP option 281
JUST= style attribute 289
MISSING option 100
NOPRINT option 284
NOZERO option 288–289
ORDER= option 281, 366
PRELOADFMT option 365–366
superscripts and 340
DEFINEDATA method 120, 228
DEFINEDONE method 120, 228
DELETE method 127–128
DELETE option, INDEX statement (DATASETS) 205
DELETE procedure 211
DELETE statement, DATASETS procedure 211
DELETEFUNC statement, FCMP procedure 484–485
DELETESUBR statement, FCMP procedure 484
deleting
catalogs 211
data sets 211
Excel sheets 7
DelGobbo, Vince 333, 335
DELIMITER= option
CSV tagset 31
FILE statement 28–29
INFILE statement 21, 23–24
DISPLAY option, DEFINE statement (REPORT) 281
DISTINCT function 93
DLL (Dynamic Link Library) 470–471
DLM= option
FILE statement 28–29
FILE statement 21, 23–24
DM statement
about 466
additional information 452
executing commands 445, 451–452
quotations marks and 79
WRTFSAVE option 440
DMOPTLOAD command 445, 452
DMOPTSAVE command 445, 452
%DO loop
EXPORT procedure and 335
semicolons and 404
usage example 16
DO loops
compound 178
key index lookups 225
LAG function in 109
MIDPOINTS option and 272
OUTPUT statement in 64
principles of 176–180
special forms 178–180
DO UNTIL loop
breaking up data sets 127
eliminating duplicate observations 95
FINDC function and 159
HASH object example 228–229
key index lookups 224
stepping through hash tables 123, 126
variable information functions example 153
exception reporting 410–415
controlling data validations 58–60
customizing 474
controlling with macros 416–417
writing to error data sets 55–58
%EXCEPTIONS macro 416–417
EXCLUSIVE option
CLASS statement (MEANS) 235, 369–370
CLASS statement (SUMMARY) 235, 369–370
CLASS statement (TABULATE) 367–368
MEANS procedure 71, 251–252, 364
REPORT procedure 364–367
SUMMARY procedure 70, 251–252, 364
TABULATE procedure 252, 267–268, 364
EXIT command (DOS) 478
EXPAND procedure 101, 380
expanding dates 137
Explorer Options: Table Options dialog box 464–465
Explorer Options dialog box 463
Explorer window 463
EXPORT procedure
about 9
additional information 335
DATA= option 9
DBMS= option 10, 29
DELETESUBR statement 335
exporting CSV files 12–15
OUTFILE= option 9, 29
preventing export of blank sheets 15–16
REPLACE option 9–10
SHEET= statement 9
writing delimited files 29–30
exporting CSV files 12–15
expressions, evaluating
  See evaluating expressions
Extensible Markup Language (XML)
  EXCELXP tagset and 335
  MARKUP destination 34
  reading and writing to 33
  XML engine 34–35
F
F= option 9
  See FONT= option
F2C function 481
F2CC function 386–387
FCMP Function Editor 483–485
FCMP procedure
about 479
  additional information 480–481
  age measurement formula and 117
  DELETEFUNC statement 484–485
  DELETESUBR statement 484
  ENDSUB statement 480
  FUNCTION statement 386, 480, 482
  interacting with macro language 482–483
  OUTLIB= option 386, 481, 485
  passing values to functions and 384
  RETURN statement 386, 480
  SUBROUTINE statement 482
FIFO stacks 113–114
FILE= option
  ODS CSV statement 31
  ODS MARKUP statement 34
FILE statement
  DLM= option 28–29
  DLMSTR= option 28
  DSD option 28–29, 32
  EMAIL engine and 467
  LRECL= option 487
FILENAME function 423
FILENAME statement
  ATTACH= option 467
  executing OS commands 477
  FROM= option 467
  PIPE device type and 478
  SUBJECT= option 467
  TO= option 467
FILENAME window 462
  filtering missing values 382
FIND function 157
FIND method
  about 120
  hash tables referencing hash tables 130
  stepping through hash tables 122–125
  table lookup techniques 228–229
%FINDAUTOS macro 423
FINDC function 157, 159
FINDW function 157
FIPSTATE function 385
FIRST processing
  BY-group processing and 92–93, 105–107, 123
  eliminating duplicate observations 92–93
  transposing data to arrays 108
FIRST method 125, 127
FIRSTOBS= data set option 42–45, 110–111
FLOOR function 117
FLOWOVER option, INFILE statement 25
FLYOVER= attribute 79–80
FMTSEARCH= system option 394
FONT catalog 318
FONT= option
  AXIS statement 307
  FOOTNOTE statement 298
  TITLE statement 298
  FONT_FACE= attribute 266
  fonts
    building 317–318
    default selections 273
    FONT catalog and 318
    TrueType 319–320
  FONT_SIZE= attribute 266
  FONT_STYLE= attribute 266
  FONT_WEIGHT= attribute 266
  FONT_WIDTH= attribute 266
  BCOLOR= option 298–299
  BOLD option 298
  #BYVAL option 475–476
  #BYVAR option 475–476
  COLOR= option 298
  FONT= option 298
  HEIGHT= option 298
  ITALIC option 298
  JUSTIFY= option 298
  LINK= option 347
  ODS supported options 298
  PAGEOF formatting sequence 338
  UNDERLINE option 298
FORCE option
  APPEND statement (DATASETS) 90
  SORT procedure 190
FOREGROUND= attribute 266
FORMAT catalog entry type 393
format libraries
  about 393
  concatenating format catalogs 394–395
format libraries (continued)
  saving formats permanently 393–394
  searching for formats 394
format modifiers
  about 18
  checking date strings 53
  for INPUT statement 18–20, 22
FORMAT procedure
  CNTLIN= option 220, 227, 390
  CNTLOUT= option 391
  INVALID statement 141, 352, 390
  LIBRARY= option 393–394
  PICTURE statement 370–377, 390
  REGEXPE option 384
  table lookup techniques 219–221
  VALUE statement 270, 352, 377–388, 381, 384, 390
FORMAT statement
  in DATA steps 20
  reordering variables on PDV and 202
  SUMMARY procedure 237
  TABULATE procedure 381
  variable information functions and 152
format translations
  about 382
  filtering missing values 382
  handling text with numeric values 383–384
  mapping overlapping ranges 383
  passing values into function 384–388
FORMATC catalog entry type 393
formats
  See also inline formatting
  ANYDATE informats and 388–390
  building from data sets 390–392
  checking date strings with 53–54
  conditionally assigning 354
  controlling order with NOTSORTED option 381
  displaying small probability values 392–393
  multilabel 377–380
  passing values into 384–388
  picture 370–377
  preloaded 72, 364–370
  saving in catalogs 393
  saving permanently 393–394
  searching for 394
  table lookup techniques 214, 219–221
  formulas, storing as data values 415
  fractional values, picture formats 373–374
FRAME option, LEGEND statement 308
FREQ procedure
  about 277
  BY variables and 475
  %DOPROCESS macro and 329
  duplicate observations and 93
  graphics and 323
  NLEVELS option 278
  ODS OUTPUT statement 329
  OUTPUT statement 277–278
  QNUM function and 387
  SPARSE option 73
  TABLE statement 73, 93, 100, 236, 277–279, 323
Friendly, Michael 156, 314
FROM CONNECTION phrase (SQL) 32
FROM= option, FILENAME statement 467
FROM statement, SQL procedure 93
FTEXT= graphics option
  migrating text 273
  setting fonts 274, 300–301, 319
  UNIVARIATE procedure and 302
FTITLE= graphics option 302
Function Editor (FCMP) 483–484
FUNCTION statement, FCMP procedure 386, 480, 482
functions
  See also specific functions
  alternative 154–163
  ANY family of 144–145
  collecting setting values through 422–424
  comparison 145–147
  concatenation 147
  counting 155
  for age calculations 116–117, 419
  interacting with macro language 482–483
  macro 417–419
  NOT family of 144–145
  passing values into 384–388
  powerful and flexible 154–163
  removing 484–485
  storing and accessing 481–482
  underutilized 143–165
  user-defined 386–387, 479–485
  variable information 148–154
  viewing definitions 483–484
fuzzy merges 171
G
GCHART procedure 272, 348
Gebhart, Eric 333
$GENDERU. format 365
GEOMEAN function 156
GET operator 47
%GETDATANAME macro 400
%GETFUNC macro 472
%GETGLOBAL macro 440
GETNAMES option, LIBNAME statement 7
GETNAMES= statement, IMPORT procedure 10–11, 13, 443
GETOFTION function 110, 422–423, 469
GDFONT procedure 317–318
GIF files 348
GKPI procedure 320–322
GLM procedure 100
%GLOBAL statement 399, 401
GOPTIONS procedure 300–302, 319
GPLOT procedure 314–316
%GRABDRIVE macro 471
%GRABPATHNAME macro function 470
graphics elements, linking 348–350
graphics fonts, building 317–318
Graphics Stream File (GSF) 301
GROUP option, DEFINE statement (REPORT) 281
GROUPINTERNAL option
  CLASS statement (MEANS) 235, 237
  CLASS statement (SUMMARY) 235, 237
GSF (Graphics Stream File) 301
GFSMODE= graphics option 301
GSFNAME= graphics option 271, 300–301
GSUBMIT command 461–466
GUESINGROWS= statement, IMPORT procedure 10–12
H
hard coding issues 415–417
HASH object
  about 94, 118
  additional information 118–119
  defining and loading 120–121
  determining unique keys 94–95
  eliminating duplicate observations 94–96
  many-to-many merges and 171
simple sorts using 120–121
table lookup techniques 227–229
hash sign (#) 350–351, 430–431
hash tables
about 118
creating 119
key indexing and 223–227
referencing hash tables 128–130
stepping through 121–126
table lookup techniques 214, 227–229
updating master data sets 130–131
HASHEXP: constructor 119
Haworth, Lauren 258
HAXIS= option, PLOT statement (BOXPLOT) 315
HBBOUND function 182–183
HBBULLET statement, GKPI procedure 320–321
HEADER option, LIBNAME statement 7
HEIGHT= option
AXIS statement 307
FOOTNOTE statement 298
SYMBOL statement 304
TITLE statement 298
Henderson, Don 176, 474
HEX. format 143
HEX16. format 169
hexadecimal number conversions 143
hiding macro code 426–427
hierarchy of operators 45–46
HISTOGRAM statement, UNIVARIATE procedure
about 270
MIDPOINTS option 272
OUTHISTOGRAM= option 273
histograms
linking to reports 348–349
UNIVARIATE procedure and 270, 272–273
HITER object
about 118
accessing hash tables 119
stepping through hash tables 122, 125–126
HPOS graphics option 402
HTEXT= graphics option 300, 302
HTITLE= graphics option 302
HTML anchor tags 295
HTML destination
about 332
ASIS style attribute and 358
linking graphics elements 348
HTML3 destination 332
HTML option, VBAR statement (GCHART) 348
HTML4 tagset 332
HTML_LEGEND option, VBAR statement (GCHART) 348
Huang, Charlie 462
Huntley, Scott 357
hyperlinks
about 345
creating internal links 350–351
linking graphics elements 348–350
style overrides and 345–347
hyphen (–) 438–441

I

I= option, SYMBOL statement 315–316
ID statement
PRINT procedure 291–292
TRANSPOSE procedure 62, 153, 199
UNIVARIATE procedure 327
IDGROUP option, OUTPUT statement 61, 243–244
IDXNAME data set option 206–207
IDXWHERE data set option 206–207
IF statement
CALL SYMPUTX routine comparison 163, 179–180
conditionally assigning formats 354
DLM option and 23
logical and comparison operators in 48–49
MIN and MAX operator and 50–51
negative values and 51
table lookup techniques 214–216
IFC function 156–158
IFN function 156–157
implicit arrays 183–184
IMPORT procedure
about 9
data source statements 10–12
DATAROW statement 10
DBMS= option 10, 12
GETNAMES statement 10–11, 13, 443
GUESSINGROWS= statement 10–12
importing CSV files 12–15
MIXED= statement 11–12
NAMEROW= statement 12
RANGE= statement 10, 17
REPLACE option 10
SCANTEXT statement 10
SHEET= statement 10
STARTROW= statement 12
TEXTSIZE statement 10
working with named ranges 16–17
importing CSV files 12–15
IN comparison operator
DATA steps and 47, 430
in macro language 430–433
SQL procedure and 47, 430
INAGE. informat 383
%INCLUDE statement 406, 462
indentations 342–343
INDEX function
about 157, 159
ANY family of functions and 144
mixed dates example 390
semicolumns and 163
INDEX statement, DATASETS procedure
about 222
CREATE option 204
DELETE option 205
INDEXC function 157
indexes
about 193, 202–204
BY statement 222
caveats and considerations 207
composite 203, 206
KEY= option, SET statement 203, 222–223
simple 203–205
table lookup techniques 214, 221–223
INDEXW function 157
indicator bars and dials 320–322
INDNAME= option, SET statement 172, 174–175
inequalities, compound 49–50
INFILE, automatic variable 17
INFILE statement
DELMITIER option 21
DLM= option 21, 23–24
DLMOPT option 21
DLMSTR= option 21, 23–24, 28–29
DSD option 21
FLOWOVER option 25
LENGTH= option 26
MISSOVER option 25, 27–28
TRUNCOVER option 25–28
INFMT catalog entry type 393
INFMTC catalog entry type 393
INFORMAT statement
  in DATA steps 20
  reordering variables on PDV and 202
information sources (macro)
  about 420
  accessing metadata for data sets 424–426
  DICTIONARY tables 420–421
  SASHELP views 420–421
informs
  saving in catalogs 393
  user-defined 140–141
initialization options 438–441
  -INITSTMT initialization option 440–441, 444
inline formatting
  changing text attributes 341–342
  controlling indentations 342–343
  controlling line breaks 342–343
  controlling spacing 342–343
  dagger symbol 340–341
  escape character sequences and page X of Y 338–339
  subscripts 340–341
  superscripts 340–341
inline style modifiers 341–342
INPUT function
  about 139
  checking date strings with formats 53–54
  datetime values and 390
  key indexing and 224, 226
  %SYSFUNC function and 138
  table lookup techniques 221
  variable conversions 138–142
INPUT statement
  about 17
  additional information 17
  controlling delimited input 20–24
  format modifiers for 18–20, 22
  reading variable-length records 24–28
INPUTC function 141
INPUTN function
  additional information 142
  automatic dates and 138
  execution considerations 141
  %SYSFUNC function and 139
  INSERT option, CONFIG.CFG file 447
Insert String dialog box 458–459
INSET statement, UNIVARIATE procedure 270–271, 273
INSIDE option, LEGEND statement 308
INTCK function
  about 116, 132
  additional information 132
  alignment options 134–136
  automatic dates 137
  shift operators 132–134
  START function and 484
  internal links, creating 350–351
INTERPOL= option, SYMBOL statement 304, 315–316
INTERSECT operator (SQL) 93
interval multipliers 132–133
INTNX function
  about 132
  additional information 132
  alignment options 133–135
  automatic dates 136–138
  interval multipliers 132–133
  shift operators 132–134
  START function and 484
  variable conversion example 142
INTO : clause, SELECT statement (SQL) 410
INVALUE statement, FORMAT procedure
  creating formats 390
  creating informats 141, 390
  traffic lighting and 352
IS MISSING operator 83–84
IS NULL operator 84
ITALIC option
  FOOTNOTE statement 298
  TITLE statement 298
J

J= option
  See JUSTIFY= option
Java object 118
JAVAIMG device 321
joins and merges
  about 165
  BY variable attribute consistency and 166–169
  fuzzy 171
  in DATA steps 165–171, 216–218
  repeating BY variables 170–171
  table lookup techniques 214
  variables in common 169–170
Jordan, Mark 468
JUST= style attribute 289
JUSTIFY= option
  about 319
  FOOTNOTE statement 298
  TITLE statement 298
K
KEDYDEF command 466
KEEP= data set option
  about 39, 42–43
  duplicate observations and 93
  KEEP statement and 8, 42
  reordering variables on PDV 201
  SORT procedure and 191
  variable values and 30
KEEP statement
  KEEP= data set option and 8, 42
  reordering variables on PDV 201–202
  shorthand variable naming and 73–74
  key indexing (direct addressing) 214, 223–227
  short-hand variable naming and 73–74
  KEYLEVEL= option, CLASS statement (UNIVARIATE) 274
  KEYS window 445, 462, 466–467
  KEYWORD statement, TABULATE procedure 265
  Kill option, DATASETS procedure 211
  King, John 179
  KMF files 457
  KPI (Key Performance Indicator) 320–322
L
LABEL= option
  AXIS statement 307
  LEGEND statement 308
  TABLE statement (TABULATE) 266
LAG function 108–109
Langston, Rick 479
LARGEST function 147–148
LAST. processing
  BY-group processing and 92–93, 105–107
eliminating duplicate observations 92–93
transposing data to arrays 108
LASTMY macro function 142
LASTPAGE formatting sequence 339
LBOUND function 182–183
leading blanks 163
LEFT function 140, 167
%LEFT macro function
autocall libraries and 406, 417
quotation marks and 435
removing characters from text strings 163
LEGEND= option, LEGEND statement 308
LEGEND statement
about 306
ACROSS option 308
FRAME option 308
generating box plots 315
INSIDE option 308
LABEL= option 308
LEGEND= option 308
NOLEGEND option 308
OUTSIDE option 308
SHAPE= option 309
VALUE= option 308
LENGTH function 163, 385
%LENGTH macro function 101
LENGTH= option, INFILE statement 26
LENGTH statement
about 27
in joins and merges 168
reordering variables on PDV 201
RETAI option and 202
setting variable attributes 96
usage example 162
LEN= format 386
LLEN. format 386
LEVELS option, OUTPUT statement 254
LIBNAME function 208
LIBNAME statement
CLEAR option 8
data access engines and 4–8
GETNAMES option 7
HEADER option 7
MIXED option 7
PASSWORD option 6
SCAN TEXT option 7
USER option 6
VER option 7
working with named ranges 16–17
XMLFILERE= option 34
LIBNAME window 462
LIBRARY= option, FORMAT procedure 393–394
LIFO stacks 113
LIKE operator 83–85
line breaks 342–343
LINE= option, SYMBOL statement 304
LINE statement, REPORT procedure
aliases in 281
changing text attributes 342
conditionally executing 290–291
STYLE= option 285–287
superscripts and 340
LINK= option
FOOTNOTE statement 347
TITLE statement 347, 351
LIST style input 18, 20
LISTING destination
format considerations 264
HTML anchor tags and 295
linking graphic elements and 348
RTS= option and 265
STYLE= option and 285
LOCAL macro function 398–401
LOG window 462
Logger object 118
logical operators in assignment statements 47–49
logo symbol 318
look-ahead technique
additional information 105, 110
double SET statement and 111
MERGE statement and 110
SET statement and 174
look-back technique
additional information 105
LAG function and 108–109
SET statement and 111–113, 174
LRECL= option, FILE statement 487
M
~m sequence code 342–343
macro abbreviations for Enhanced Editor 456–460
macro functions 417–419
See also specific macro functions
macro information sources
about 420
accessing metadata for data sets 424–426
DICTIONARY tables 420–421
SASHELP views 420–421
macro language
avoiding macro variable collisions 398–400
building macro variables 400–403
#BYVAL option and 475
#BYVAR option and 475
comments and 410, 418
controlling exception reporting with macros 58–60
debugging considerations 210, 403–405, 433
executing specific versions 427–430
functions interacting with 482–483
generalized programs and 403–406
IN operator 430–433
macro information sources 420–429
macro libraries and 406–409
metadata-driven programs and 409–415
MFILE system option and 415
missing values and 101
quotation marks and 434–435, 475
replacing hard coding with 415–417
security and protection considerations 426–430
writing macro functions 417–419
macro libraries 406–409
%MACRO statement
MINDELIMITER= system option and 431
processing overview 407
SECURE option 427
SOURCE option 426–427
STORE option 408
macro variables
avoiding collisions 398–400
building 400–403
building list of 402–403
missing values and 101
quotation marks and 80
resetting graphics options 402
MAJOR= option, AXIS statement 307
MAKELIST macro 425–426
MAKETEMPWORK macro 466
MAKEXLS macro 16
Index

mapping overlapping ranges 383
MARKUP destination about 33–34
EXCELPX tagset 333
linking reports from 348
MATCH_ALL option, ODS OUTPUT statement 330–332
MAUTOLOCDISPLAY system option 408
MAUTOSOURCE system option 407
MAX function 50, 147–148
MAX operator 50–51, 86
MAX statistic 241–243
MAXID option, OUTPUT statement 241–243
maximum values
finding 147–148
MAX function 50, 147–148
MAX operator 50–51, 86
MAXWT_B. format 353
MAXWT_F. format 353
MDYAMPM. informat 389
MEAN= option, OUTPUT statement 240–241
MEANS procedure
about 233–234
CHARTYPE option 247–248
CLASS statement 78, 106, 191–192, 234–238, 255, 404
CLASSDATA= option 70–71, 251–252
COMPLETETYPES option 70, 253, 369–370
EXCLUSIVE option 71, 251–252, 364
generalizing programs example 404
identifying extremes 241–245
naming output variables 238–240
NWAY option 247, 276
ORDER= option 77–79
OUTPUT statement 238–245, 254
preloaded formats and 72, 364, 369–370
THREADS system option and 195
transposing data and 61
_TYPE_ automatic variable and 246–248
WAYS statement 250–251
VAR statement 240
WAYS statement 249–250
MEMTYPE= option, DATASETS procedure 211
%MEND statement 407
MERGE statement
double SET statement and 111, 176, 218–219
in joins and merges 168
look-ahead technique and 110
MERGENOBY= system option and 441
repeating BY variables and 170
table lookup techniques 216–218
MERGENOBY= system option 110, 441–442
merges and joins
See joins and merges
metadata
about 409
accessing for data sets 424–426
controlling data validations 410–415
macro language and 409–415
processing across data sets 409–410
sort considerations 193–194
sources of information for 410
methods
about 119
accessing within objects 119–120
dot notation and 120
return codes 121, 126
MFILE system option 433
MI procedure 101
MIDPOINTS option, HISTOGRAM statement (UNIVARIATE) 272
MIN function 50, 147–148
MIN operator 50–51, 86
MIN statistic 241–243
MINDELIMITER= system option 431–432
MINID option, OUTPUT statement 241–243
minimum values
finding 147–148
MIN function 50, 147–148
MIN operator 50–51, 86
MINOPERATOR system option 430–433
MINOR= option, AXIS statement 307
MISSDATE. format 382
MISSING function
about 99–100
checking for missing date values 55
negation of 51
MISSING method 120
MISSING option
CLASS statement (MEANS) 100, 234–236
CLASS statement (SUMMARY) 100, 234–236
DEFINE statement (REPORT) 100
TABLE statement (FREQ) 100
MISSING routine
See CALL MISSING routine
MISSING statement 97
MISSING system option 98
missing values
additional information 97
CALL MISSING routine 96, 100
checking for missing dates 54–55
classification variables 100
CMISS function and 99–100
filtering 382
imputing 101
macro variables and 101
MISSING function and 51, 55, 99–100
MISSING system option 98
NMISS function and 99–100
numeric 383–384
replacing with zero 51
special 97–98
SUM function and 114
MISSOVER option, INFILE statement 25, 27–28
MISSTEXT= option, TABLE statement (TABULATE) 262
MIXED option, LIBNAME statement 7
MIXED procedure 314
MIXED= statement, IMPORT procedure 11–12
MLF (multilabel) formats 377–380
MLF option
CLASS statement (MEANS) 235
CLASS statement (SUMMARY) 235
MLF option, CLASS statement (TABULATE) 378
MLOGIC system option 422, 433
MLOGICNEST system option 433
MMDDY. format 53
MOD function 113–114
MODIFY statement
DATASETS procedure 76, 222
hash tables updating master data sets 130
MODULEC function 470
MODULEN function 470
MONNAME. format 142, 372
MONTH function 46–47
MONTHABB. format 372
MONTHNAME. format 372
moving average calculation 107, 113–114, 378–380
MPRINT system option 422, 427, 433
MPRINTNEST system option 433
MSLEVEL= system option 203, 205
MSOFFICE2k destination 332
MSTORED system option 408
MULT= option, PICTURE statement (FORMAT) 374–377
multilabel (MLF) formats 377–380
MULTILABEL option, VALUE statement (FORMAT) 377–378
MYDATT. format 372
%MYMEANS macro 404
MZERO. format 262
_N_ automatic variable 112, 151
%N directive 373
N= option, OUTPUT statement 240–241
N statistic 240–241, 288
NAME= option, HBULLET statement (GKPI) 321
named ranges 16–17, 74–75
naming
  compound variable names 281
  output variables 238–240
  report items in compute block 280–281
  shorthand variables 75–76
Nelson, Rob 357
nesting
  dates 288–289
  formats 383
  macros 398–400
  tables 260–261
  NEW_ keyword 128
NEW LIBRARY window 4
NEXT method 126–130
%NEXTDOG macro function 419
NLEVELS option, FREQ procedure 278
NMISS function 99–100
-NOAWSMENU initialization option 449
NOBYLINE system option 245
NODUPKEY option, SORT procedure
  eliminating duplicates example 92
  filling sparse data example 66
  joins and merges example 169–170
  key indexing and 223
  NODUPREC option and 187
  simple sort example 121
NODUPLICATES option, SORT procedure 91
NODUPREC option, SORT procedure 186–187, 190
NOEQUALS option, SORT procedure 190
NOFMTEERR system option 53
NOLEGEND option, LEGEND statement 308
NOLIST option, DATASETS procedure 211, 222
NOMAUTOLOCDISPLAY system option 408
NOMCOMPILE system option 427–428
NOMINOPERATOR system option 431
NOMLOGIC system option 427
NOMPRINT system option 427
NOMREPLACE system option 427–429
NOOBS option, PRINT procedure 31
NOPRINT option, DEFINE statement (REPORT) 284
NOPRINT option, TABLE statement (FREQ) 279
normalizing data 60–64
NOSORTEQUALS system option 190
NOSYMBOLGEN system option 427
NOT operator 83–84
NOTALPHA function 145
NOTCHES option, PLOT statement (BOXPLOT) 314
NOTDIGIT function 145, 164
notes, customizing 474
NOTTHREADS system option 195
NOTSORTED option, VALUE statement (FORMAT) 270, 381
NOTXDIGIT function 143
-NOWORKINIT initialization option 441
-NOWORKTERM initialization option 441
NOXSYNC system option 478
NOXWAIT system option 478
NOZERO option, DEFINE statement (REPORT) 288–289
%NRSTR macro function 435, 465
numbered range variable lists 73–74
numeric expressions, evaluating 51–52
NUMERIC list modifier 75
numeric missing values 383–384
_NUMERIC_ variable name list 76, 99, 182
numeric variables
  FIRST. and LAST. processing 92–93, 105–107
  NMISS function and 99–100
  setting length of 81
  shorthand naming 75–76
  variable conversions and 138–142
NWAY option
  MEANS procedure 247, 276
  SUMMARY procedure 247, 276
objects
  accessing methods within 119–120
  creating and naming 119
  determining names of 326–327
  dot notation and 120
  labels and ODS OUTPUT statement 328
  OBS= data set option 42–45
%OBSCNT macro 408, 418, 465
observations
  additional information 105
  building FIFO stacks 113–114
  BY-group processing 105–107
  eliminating duplicate 90–96
  identifying extremes 241–245
  LAG function and 108–109
  look-ahead and MERGE statement 110
  look-ahead and SET statement 111
  look-back and SET statement 111–113
  processing across 105–114
  SUM statement and 114
  transposing to arrays 64, 107–108
O’Conner, Dan 357
OCTAL. format 143
ODS (Output Delivery System)
  about 297, 326
  additional information 326
  creating hyperlinks 345–351
  escape character sequences and 337–345
  graphics options and settings 300–302
  inline formatting and 337–345
  reading and writing to XML 34
  STYLE= option and 266
  title and footnote options 352–356
  useful tidbits 358–359
  writing delimited files 31
  writing reports to Excel 332–336
ODS CSV statement 31
ODS ESCAPECHAR option 427
ODS GRAPHICS statement 352–356
ODS LAYOUT statement 356–357
ODS LISTING statement 331
Index

ODS MARKUP statement
  EXCELXP tagset and 333
  FILE= option 34
  STYLE= option 336
ODS NOUSEGOPT statement 302
ODS OUTPUT statement
  creating data sets 329
  data set options and 326
  MATCH_ALL option 330–332
  object labels and 328
  PERSIST= option 330–332
ODS PDF statement 349–351, 357
ODS PROCLABEL statement 349, 351
ODS REGION statement 356
ODS RESULTS statement 358–359
ODS RTF statement 299, 338–339
ODS TRACE statement 327
ODS USEGOPT statement 302
OPEN= option, SET statement 172
operator hierarchy 45–46
OPTIONS option
  EXCELXP tagset 333
  ODS CSV statement 31
  OPTIONS procedure 300, 444–445
  OPTLOAD procedure 444–445
  OPTSAVE procedure 444–445
ORDER BY statement, SQL procedure 93
ORDER= option
  about 77–79
  AXIS statement 307
  CLASS statement 192, 235, 237–238
  CLASS statement (MEANS) 78, 235, 237–238
  CLASS statement (SUMMARY) 78, 192, 235, 237–238
  DEFINE statement (REPORT) 281, 366
  MEANS procedure 77–79
  TABULATE procedure 269–270
TITLE statement 77
ORDERED: constructor 119, 126
ORDINAL function 147–148
OS commands
  additional information 479
  data step execution 478
  global execution 477–478
  sub-session execution comments 478–479
OUT= option
  COMPARE procedure 198
  CONTENTS procedure 424–425
  OUTPUT statement (SUMMARY) 239
  SORT procedure 5
TRANSPOSE procedure 61
OUTBASE option, COMPARE procedure 198
OUTCOMP option, COMPARE procedure 198
OUTFILE= option, EXPORT procedure 9, 29
OUTHISTOGRAM= option, HISTOGRAM statement (UNIVARIATE) 273
OUTLIB= option, FCMP procedure 386, 481, 485
OUTNOEQUAL option, COMPARE procedure 198
Output Delivery System
  See ODS (Output Delivery System)
OUTPUT destination
  about 326
  creating data sets 327–329
  determining object names 326–327
  MATCH_ALL option 330–332
  NLEVELS option and 279
  PERSIST= option 330–332
OUTPUT method
  breaking up data sets 126–128
  hash tables referencing hash tables 128–130
  simple sort example 120–121
OUTPUT statement
  See also ODS OUTPUT statement
  AUTOLABEL option 239–240
  AUTONAME option 239–240
  conditionally executing 151
  ELSE statement and 55
  FREQ procedure 277–278
  IDGROUP option 61, 243–244
  in DO loops 64
  LEVELS option 254
  MAXID option 241–243
  MEAN= option 240–241
  MEANS procedure 238–245, 254
  MINID option 241–243
  N= option 240–241
  naming output variables 238–240
  PCTLPRE= option 277
  PCTLPPTS= option 277
  statistic specification 240–241
  SUMMARY procedure 238–245, 254
  UNIVARIATE procedure 276–277
WAYS option 254
output variables, naming 238–240
OUTSIDE option, LEGEND statement 308
overlapping ranges, mapping 383

P

PAGEBY statement, PRINT procedure 476
PAGEOF formatting sequence 338–339
parentheses () 119
pass-through (SQL) 32–33, 208–210
passing values as format labels 384–388
PASSWORD option, LIBNAME statement 6
password protection 41, 208–210
PATHNAME function 423, 468
PATTERN statement 317
PCTLPRE= option, OUTPUT statement (UNIVARIATE) 277
PCTLPPTS= option, OUTPUT statement (UNIVARIATE) 277
PCTZERO. format 382
PDF destination 339, 348
percent sign (%) 84–85, 434–435
percentages, calculating 262–264, 276–277
percentile statistics 245
period () 97–98
Perl regular expressions 384
PERSIST= option, ODS OUTPUT statement 330–332
physical location information 468–472
picture formats
  about 370
  date directives and 370–372
  display granularity and 376–377
  fractional values and 373–374
  preceding text and 374–376
  truncating 374
PICTURE statement, FORMAT procedure
  about 370, 390
  date directives and 370–372
  display granularity and 376–377
  fractional values and 373–374
  preceding text and 374–376
PLOT statement
  BOXPLOT procedure 314
  REG procedure 305
plot symbols 303, 318
See also SYMBOL statement
PLOTS= option, TABLE statement (FREQ) 323
PMENU procedure 462
PNG files 348
POINT= option, SET statement 172–174
DO loops and 180
look-ahead technique and 111–113
POINTLABEL option, SYMBOL statement 311
pop-up menus, adding tools to 463–465
positive values, determining 52
pound sign (#) 350–351, 430–431
POUNDS. format 376
PREFIX= option
PICTURE statement (FORMAT) 374–376
TRANSPOSE procedure 61, 67
prefix variable lists 73–74
preloaded formats about 72, 364
MEANS procedure 72, 364, 369–370
modifying report contents with 364–370
REPORT procedure and 72, 364–367
SUMMARY procedure 72, 364, 369–370
TABULATE procedure and 72, 364, 367–368
PRELOADFMT option
CLASS statement (MEANS) 235, 369
CLASS statement (SUMMARY) 235, 369
CLASS statement (TABULATE) 367–368
%PRIMARY statement 399–400
PRINT procedure about 291
BY statement 291–292
filtering missing values 382
generating table of contents 295
ID statement 291–292
NOOBS option 31
PAGEBY statement 476
reordering variables on PDV 200
STYLE= option 292–294
style overrides and 345–347
TITLE statement 245
traffic lighting and 352, 355–356
VAR statement 31, 294, 355
WHERE statement 351
%PRINTALL macro 409–410
%PRINTTI macro 465, 483
PRINTMISS option, TABLE statement (TABULATE) 367–368
PRINTTO procedure 439–440
probability plots 275, 303
probability values, displaying 392–393
PROB procedure 303
PROBPLOT statement, UNIVARIATE procedure 270, 275
process automation 198–200, 329
process control charts, generating 316–317
%PROCESS macro 329
PRXCHANGE function 384
PTCN option, TABLE statement (TABULATE) 263–264
PTCSUM option, TABLE statement (TABULATE) 263–264
pull-down menus, adding tools to 463–465
%PURGEWORK macro 429
PUT function about 139
CALL SYMPUT routine and 401
execution considerations 141
in joins and merges 167
%SYSFUNC function and 138
table lookup techniques 221
variable conversions 138–142
PUT statement conditional 29
customizing text written to logs 474
generating e-mails 467
inserting separators manually 31
variable conversions 143
%PUT statement 465, 474
PUTC function 141
PUTLOG statement 474
PUTN function
automatic dates and 138
execution considerations 141
%SYSFUNC function and 139, 142, 371
PVALUE. format 392–393
PW data set option 41
PWENCODE procedure 208–210
PWREQ data set option 41
Q
%QLEFT macro function 163, 435
QNUM function 387, 479–481
QQPLOT statement, UNIVARIATE procedure 270, 276
%QSCAN macro function 423, 470
%QSYSFUNC macro function 434
QTR function 288
%QTRIM macro function 163, 406, 417
quantile plots (QQplots) 276, 303
QUERY command 464
question mark (?) as format modifier 18
CONTAINS operator and 84
quotation marks (") about 79–81
DSD option and 21
macro language and 434–435, 475
%QUOTE macro function 427
R
%RAND_WO macro 173
RANGE. format 393
RANGE= statement, IMPORT procedure 10, 17
RANUNI function 173
READ data set option 41
reading data in variable-length records 24–28
look-ahead technique 105, 110–111
look-back technique 105, 108–109, 111–113
mixed dates and 389
to XML 33–35
with data access engines 5
REG procedure
NOLEGEND option and 308
PLOT statement 305
SAS/GRAPH support 303
REGEXPE option, FORMAT procedure 384
regular expressions (Perl) 384
$REGX. format 365–366
RENAME= data set option about 42–44, 444
appending data sets 89
RENAME statement and 42
table lookup techniques 220
RENAME function 209, 212
RENAME statement 42, 202
renaming catalogs 212
data sets 211–212
reordering case-sensitive 189
numeric strings 188–189
variables on PDV 200–202
REPEMPHY data set option 40–41
REPLACE data set option 40–41
REPLACE method 94–95, 120, 124
REPLACE option
  EXPORT procedure 9–10
  IMPORT procedure 10
Repole, Warren 430
report items 280–281
REPORT procedure
  about 280
  aligning decimal points 289–290
  CALL DEFINE routine 287–288
  COLUMN statement 281–284
  COMPLETECOLS option 365
  COMPLETEROWS option 72, 365–367
  compute block and 280–291
  consolidating columns 284–285
  CONTENTS= option 349
  EXCLUSIVE option 364–367
  indicator bars and dials 321–322
  nested dates 288–289
  preloaded formats and 72, 364–367
  style overrides and 345–347
  TABULATE procedure and 280
  THREADS system option and 195
  traffic lighting and 352, 354–355
reports
  modifying contents with preloaded formats 364–370
  writing to Excel tables 332–336
RESET= graphics option 301, 304
RETAIN statement
  reordering variables on PDV and 202
  SUM statement and 114
  table lookup techniques 220
return codes (methods) 121, 126
RETURN statement, FCMP procedure 386, 480
Rhodes, Dianne 258
Rhodes, Mike 110
rolling average calculation 107, 113–114, 378–380
Rosenbloom, Mary 476
ROTATE= option, AXIS statement 308
ROUND function 159–160
ROUND option, PICTURE statement (FORMAT) 372, 374
RTF destination
  issuing raw RTF specific commands 344–345
  LASTPAGE formatting sequence 339
  linking reports from 348
  PAGEOF formatting sequence 338–339
  THISPAGE formatting sequence 339
RTF file format 485–487
-RTFCOLOR initialization option 440
RTS= option, TABLE statement (TABULATE) 265–266
RUN statement 20
RUN_MACRO function 482
S
SAME operator 83, 85
SAME operator 384
SAS/ACCESS engine 4, 6
SAS/AF application 449
SAS/GRAPH application
  about 297, 303, 313–314
  annotate facility 273, 309–311
  building indicator bars and dials 320–322
  changing text lines 319
  title/footnote options 298–300
  UNIVARIATE procedure 270, 273
SAS/QC application 303, 314, 316–317
SAS/STAT application 303, 314
SASAUTOS= system option
  automatic libraries and 407, 423
  changing SASAUTOS location 447–448
  saving system options and 444–445
SAS_EXECPFILENAME environmental variable 469
SAS_EXECPATH environmental variable 469–470
SASHELP views
  additional information 8
  attributes of data sets and 424
  list of 420–421
  recovering physical location information 468–469
SASHELP.VALLOCPT view 420–422
SASHELP.VCATALG view 420
SASHELP.VCFORMAT view 420
SASHELP.VCOLUMN view 151, 420
SASHELP.VDCTNRY view 420
SASHELP.VENGINE view 420
SASHELP.VEXTFL view 420, 469
SASHELP.VFORMAT view 420
SASHELP.VFUNC view 483
SASHELP.VGOPT view 420, 422
SASHELP.VINDEX view 420
SASHELP.VLIBNAM view 420, 468–469
SASHELP.VMACRO view 420
SASHELP.VMEMBER view 421
SASHELP.VOPTIONS view 421–422
SASHELP.VSACCES view 421
SASHELP.VSCATLG view 421
SASHELP.VTABLE view 421
SASHELP.VSTABLE view 421
SASHELP.VSTABVW view 421
SASHELP.VSTYLE view 421
SASHELP.VSVIEW view 421
SASHELP.VTITLE view 421
SASHELP.VVVIEW view 421
-SASINITIALFOLDER initialization option 439, 448
SASMSTORE= system option 408–409
!SASROOT directory 446
SASV9.CFG file 446
SAVE command 466
SAVEGLOBAL macro 440–441
SAVE macro 422
SECURE option, %MACRO statement 427
%SECURECODE macro 422
security considerations
  macro language and 426–430
  password protection 40, 208–210
SELECT statement
  DATA steps 215, 421
  SQL procedure 202, 410, 421
**Index**

sparse data

about 65
CLASSDATA= option and 70–71
COMPLETETYPES option and 70
double transpose 67–69
known template of rows 65–66
preloaded formats and 72
SPARSE option and 73
SQL procedure
CASE statement 215
CONNECT statement 32, 210
CREATE INDEX statement 204
creating indexes 203–205, 221
DESCRIBE statement 421
DICTIONARY tables and 8, 421
DISCONNECT statement 32
duplicate observations and 93
FROM statement 93
IN comparison operator and 47, 430
JOIN operations 218
ORDER BY statement 93
pass-throughs and 32–33, 208–210
SELECT statement 202, 410, 421
sort considerations 193
THREDS system option and 195
WHERE clause 82–83
START function 484–485
START option, ODS LAYOUT statement 356
STARTROW= statement, IMPORT procedure 12
STDIZE procedure 101
STOP statement 121, 131
/STORE option, %MACRO statement 408
stored compiled macro libraries 406, 408
storing formulas as data values 415
functions 481–482
%STR macro function 101, 435
strings
See text strings
STRIP function 163–164
STUDYDT. format 392
style attributes
about 335–336
CALL DEFINE routine and 287–288
changing for text 341–342
PRINT procedure and 292–294
style modifiers 341–342
STYLE= option
CLASS statement (TABULATE) 265
CLASSLEV statement (TABULATE) 266, 351
creating links 345–347
LINE statement (REPORT) 285–287
ODS MARKUP statement 336
PRINT procedure 292–294
TABLE statement (TABULATE) 265–266, 353
VAR statement (PRINT) 355
VAR statement (TABULATE) 267
SUBJECT= option, FILENAME statement 467
SUBROUTINE statement, FCMP procedure 482
subsctipts 340–341
subsets
CLASSDATA= option and 251–252
EXCLUSIVE option and 251–252
LEVELS option and 254
percentiles creating 245
TYPES statement and 250–251

semicolons (;)

%DO blocks and 404
INDEX function and 163
troubleshooting missing 40
sending e-mails 467–468

-SET keyword 447

SET statement
about 172
breaking up data sets example 127
double 111, 175–176, 214, 218–219
END= option 111, 128, 172, 175, 177, 245
HASH objects and 228–229
INDSNAME= option 172, 174–175
KEEP= data set option and 42
key index lookups 225
KEY= option 172, 203, 222
look-ahead technique and 111
look-back technique and 111–113
NOBS= option 172–174, 180
OPEN= option 172
POINT= option 111–113, 172–174, 180
reordering variables on PDV and 201
simple sort example 120–121
UNIQUE option 172
SETINIT procedure 5
SHAPE= option, LEGEND statement 309
SHEET= statement
EXPORT procedure 9
IMPORT procedure 10
SHEET_INTERVAL option, EXCELXP tagset 334
sheets
See Excel sheets and workbooks

SHEWART procedure 303, 314, 316–317
shift operators 132–134
shorthand variable lists 73–76
SHOWDEC. format 374
%SHOWSTYLES macro 336
SHOWVAL. format 373
SIGN function 52
slash (/) 239
%SLIDER macro 322
SMALLEST function 147–148
SORT procedure
BY statement 121
data set options and 190–191
DESCENDING option 234
duplicate observations and 91–92
DUPOUT= option 187–188
EQUALS option 190
FORCE option 190
metadata sort information 193–194
NODUPKEY option 66, 92, 121, 169–170, 187, 223
NODUPLOCATE option 91
NODUPREC option 186–187, 190
NOEQUALS option 190
OUT= option 5
simple sort example 120–121
sort order considerations 191–193
SORTSEQ option 188–189
table lookup techniques 217
tagsort option 121, 188
THREDS system option and 195
SORTEDBY data set option 194
SORTEQUALS system option 190
SORTSEQ option, SORT procedure 188–189
SOUNDEX function 85–86, 145
sounds like operator 85–86
SOURCE catalog entry 470–471
SOURCE option, %MACRO statement 426–427
spacing 342–343

See Excel sheets and workbooks

SHEWART procedure 303, 314, 316–317
shift operators 132–134
shorthand variable lists 73–76
SHOWDEC. format 374
%SHOWSTYLES macro 336
SHOWVAL. format 373
SIGN function 52
slash (/) 239
%SLIDER macro 322
SMALLEST function 147–148
SORT procedure
BY statement 121
data set options and 190–191
DESCENDING option 234
duplicate observations and 91–92
DUPOUT= option 187–188
EQUALS option 190
FORCE option 190
metadata sort information 193–194
NODUPKEY option 66, 92, 121, 169–170, 187, 223
NODUPLOCATE option 91
NODUPREC option 186–187, 190
NOEQUALS option 190
OUT= option 5
simple sort example 120–121
sort order considerations 191–193
SORTSEQ option 188–189
table lookup techniques 217
tagsort option 121, 188
THREDS system option and 195
SORTEDBY data set option 194
SORTEQUALS system option 190
SORTSEQ option, SORT procedure 188–189
SOUNDEX function 85–86, 145
sounds like operator 85–86
SOURCE catalog entry 470–471
SOURCE option, %MACRO statement 426–427
spacing 342–343
sparse data

about 65
CLASSDATA= option and 70–71
COMPLETETYPES option and 70
double transpose 67–69
known template of rows 65–66
preloaded formats and 72
SPARSE option and 73
SQL procedure
CASE statement 215
CONNECT statement 32, 210
CREATE INDEX statement 204
creating indexes 203–205, 221
DESCRIBE statement 421
DICTIONARY tables and 8, 421
DISCONNECT statement 32
duplicate observations and 93
FROM statement 93
IN comparison operator and 47, 430
JOIN operations 218
ORDER BY statement 93
pass-throughs and 32–33, 208–210
SELECT statement 202, 410, 421
sort considerations 193
THREDS system option and 195
WHERE clause 82–83
START function 484–485
START option, ODS LAYOUT statement 356
STARTROW= statement, IMPORT procedure 12
STDIZE procedure 101
STOP statement 121, 131
/STORE option, %MACRO statement 408
stored compiled macro libraries 406, 408
storing formulas as data values 415
functions 481–482
%STR macro function 101, 435
strings
See text strings
STRIP function 163–164
STUDYDT. format 392
style attributes
about 335–336
CALL DEFINE routine and 287–288
changing for text 341–342
PRINT procedure and 292–294
style modifiers 341–342
STYLE= option
CLASS statement (TABULATE) 265
CLASSLEV statement (TABULATE) 266, 351
creating links 345–347
LINE statement (REPORT) 285–287
ODS MARKUP statement 336
PRINT procedure 292–294
TABLE statement (TABULATE) 265–266, 353
VAR statement (PRINT) 355
VAR statement (TABULATE) 267
SUBJECT= option, FILENAME statement 467
SUBROUTINE statement, FCMP procedure 482
subsctipts 340–341
subsets
CLASSDATA= option and 251–252
EXCLUSIVE option and 251–252
LEVELS option and 254
percentiles creating 245
TYPES statement and 250–251
subsets (continued)  
WAYS option and 254  
WAYS statement and 249  
subsetting IF statements 87  
SUBSTR function  
about 161  
checking date strings example 54  
conditionally executing 158  
manipulating dates 480  
variable information functions and 154  
SUM function 114  
SUM statement 114  
SUMMARY procedure  
about 233–234  
CHARTYPE option 247–248  
CLASS statement 78, 100, 191–192, 234–238, 255  
CLASSDATA= option 70, 251–252  
COMPLETETYPES option 70, 253, 369–370  
EXCLUSIVE option 70, 251–252, 364  
FORMAT statement 237  
identifying extremes 241–245  
naming output variables 238–245  
NWAY option 247, 276  
OUTPUT statement 238–245, 254  
preloaded formats and 72, 364, 369–370  
shorthand variable naming and 75–76  
THREADS system option and 195  
TRANSPOSING date and 61  
_TYPE_ automatic variable and 246–248  
TYPES statement 250–251  
VAR statement 76  
WAYS statement 249–250  
sunflower symbol 318  
%SUPERQ macro function 210  
superscripts 340–341  
SYMBOL statement  
BWIDTH= option 316  
COLOR= option 304, 316  
generating box plots 314–315  
GLOT procedure and 315–316  
HEIGHT= option 304  
I= option 315–316  
INTERPOL= option 304, 315–316  
LINE= option 304  
POINTLABEL option 311  
probability plots and 275, 303  
quantile plots and 303  
UNIVARIATE procedure and 273  
VALUE= option 304, 316  
WIDTH= option 304  
SYMBOLGEN system option 210, 422, 433  
SYMBOLLEGEND option, PLOT statement (BOXPLOT) 314  
%SYMEXIST macro function 419  
SYMGET function 210  
%SYMP. format 365, 367  
SYMPUT routine 401–402  
SYMPUTX routine  
See CALL SYMPUTX routine 482  
%SYSCALL statement 482  
%SYSEXEC macro function 466, 477  
%SYSFUNC macro function  
about 418, 482  
accessing metadata of data sets 425  
COUNTW function and 155  
FILENAME function and 423  
IFC function and 157  
IFN function and 157  
INPUT function and 138  
INPUTN function and 139  
INTNX function and 137–138  
PUT function and 138  
PUTN function and 139, 142, 371  
quotation marks and 434–435  
%SYSGET macro function 448, 470  
-SYSIN initialization option 439–440  
SYSIN system option 469  
%SYSMACDELETE statement 429  
SYSM59G function 209  
%SYSPARAM automatic macro variable 439  
-SYSPARM initialization option 439  
%SYSRCE macro function 223  
SYSTASK COMMAND statement 477–479  
SYSTEM function 478  
system options  
See also specific options  
about 39, 438  
additional information 444  
data processing options 441–444  
initialization options 438–441  
macro language and 422–424  
saving 444–445  
T  
table lookup techniques  
about 213–214  
array processing 214  
BY statement 216, 222  
direct addressing 214, 223–227  
double SET statements 214, 218–219  
format-driven 214, 219–221  
hash tables 214, 227–229  
IF statements 214–216  
indexes and 214, 221–223  
joins and merges 214, 216–218  
key indexing 214, 223–227  
table of contents, generating 295  
TABLE statement, FREQ procedure  
about 93, 277–278  
CHISQ option 278, 323  
classification variables and 236  
MISSING option 100  
NOPRINT option 279  
PLOTS= option 323  
SPARE option 73  
TABLE statement, TABULATE procedure  
about 258–259  
BOX= option 261, 265  
combination of elements 261–262  
collapsed elements 260  
LABEL= option 266  
MISSTEXT= option 262  
nested elements 260–261  
PCTN option 263–264  
PRINTMISS option 367–368  
PTCSUM option 263–264  
RTS= option 265–266  
singular elements 259–260  
STYLE= option 265–266, 353  
tables  
building from CSV files 13–15  
collapsed elements 260  
dimension components of 259  
hash 118–119  
nested 260–261  
writing reports to 332–336  
TABULATE procedure  
about 258–262  
additional information 258, 270
calculating percentages 262–264
CLASS statement 235, 258, 265, 367–368, 378
CLASSDATA= option 70, 252, 267–268
CLASSLEV statement 265–266, 351
EXCLUSIVE option 252, 267–268, 364
FORMAT statement 381
KEYLABEL statement 262
KEYWORD statement 265
ORDER= option 269–270
preloaded formats and 72, 364, 367–368
REPORT procedure and 280
style overrides and 345–347
TABLE statement 258–266, 353, 367–368
THREADS system option and 195
traffic lighting and 352–353
VAR statement 235, 258, 265, 267
TAGSORT option, SORT procedure 121, 188
TARGET= option, HBULLET statement (GKPI) 321
TARGETDEVICE= graphics option 301
 temporary arrays 181
_TEMPORARY_ keyword 107, 181
temporary variables
FIRST. and LAST. processing 92–93, 105–107
indexes and 222–223
_TERMSTMT initialization option 440–441, 444
TEXT= option, ODS PDF statement 357
text strings
aligning across rows 341
changing attributes of 341–342
checking date strings with formats 53–54
handling with numeric values 383–384
marking blocks of in Enhanced Editor 455
migrating 273
removing characters from 163–165
reordering numeric 188–189
splitting lines of 319
text substitution (term) 405
TEXTSIZE statement, IMPORT procedure 10
THISPAGE formatting sequence 339
THREADS system option 194–195
tilde (~)
as escape character 337
as format modifier 18, 22
TIME function 385
time values 371–373
TITLE statement
BCOLOR= option 298–299
BOLD option 298
#BYLINE option 476
#BYVAL option 245, 338–339, 475–476
#BYVAR option 245, 338–339, 475–476
changing text attributes 341
COLOR= option 298
FONT= option 298
font selections in 273, 320
HEIGHT= option 298
ITALIC option 298
JUSTIFY= option 298
%LASTMY function and 142
LINK= option 347, 351
ODS supported options 298
ORDER= option 77
PAGEOF formatting sequence 338
raw RTF commands and 344
SAS/GRAPH support 305
UNDERLINE option 298
TITLE window 462
to arrays 63–64
to arrays 107–108
TRANSPOSE procedure and 61–63
TRANSTRN function 163, 165, 487
TRANWRD function 161–163
TRIM function 47, 163–164, 401
%TRIM macro function 163, 435
TRIMN function 163–165
TrueType fonts 319–320
transposing data
about 60–61
double transpose 67–69
to arrays 107–108
TRANSPOSE procedure and 61–63
TRANSTRN function 163, 165, 487
TRANWRD function 161–163
TRIM function 47, 163–164, 401
%TRIM macro function 163, 435
TRIMN function 163–165
TrueType fonts 319–320
transposing picture formats 374
TRUNCCOVER option, INFILE statement 25–28
~2n sequence code 342–343
_TYPE_ automatic variable
about 246–247
CHARTYPE option and 248
types statement and 250–251
WAYS statement and 249–250
TYPES statement
MEANS procedure 250–251
SUMMARY procedure 250–251

U

UNC (Universal Naming Convention) 470–472
UNDERLINE option
FOOTNOTE statement 298
TITLE statement 298
underscore (_) 10, 84–85
UNION operator (SQL) 88–90, 93
UNIQUE option, SET statement 172
UNIVARIATE procedure
about 270
ANNO= option 273
BY statement 328
CLASS statement 274, 328
FTEXT= graphics option and 302
generating presentation-quality plots 270–273
HISTOGRAM statement 270, 272
ID statement 327
identifying extremes 241
INSET statement 270–271, 273
ODS TRACE statement and 326–327
UNIVARIATE procedure (continued)
  OUTPUT destination and 327–332
  OUTPUT statement 276–277
  probability plots and 275
  PROBPLOT statement 270, 275
  QQPLOT statement 270, 276
  quantile plots and 276
  SAS/GRAPH support and 303
Universal Naming Convention (UNC) 470–472
%UNQUOTE macro function 80, 435
%UPCASE macro function 435
UPDATE statement 130
URL= style attribute 346
USER option, LIBNAME statement 6
validating data
about 52
  checking date strings 53–54
  in metadata-driven programs 410–415
VALIDVARNAME= system option 10, 442–444
VALUE= option
  AXIS statement 307
  LEGEND statement 308
  SYMBOL statement 304, 316
VALUE statement, FORMAT procedure about 390
  DEFAULT= option 384
  MULTILABEL option 377–378
  NOTSORTED option 270, 381
  traffic lighting and 352
VAR command 464
VAR statement
  MEANS procedure 404
  PRINT procedure 31, 294, 355
  shorthand variable lists and 73, 76
  SUMMARY procedure 76
  TABULATE procedure 235, 258, 265, 267
  TRANSPOSE procedure 69, 199
variable information functions 148–154
variable-length records, reading 24–28
variable names, shorthand lists 73–76
variables
  See also numeric variables
  character 75–76, 99–100, 138–142
  classification 100, 236
  converting 138–142
  environmental 447, 469–470
  macro 80, 101, 398–403
  naming in compute block 280–281
  output 238–240
  shorthand 73–76
  temporary 92–93, 105–107, 222–223
VARNUM option, CONTENTS procedure 74, 200
VARRAY function 149
VARRAYX function 149
VARYING function 425
SVARYING informat 27
SVARYING informat 26–28
VAXIS= option, PLOT statement (BOXPLOT) 315
VBAR statement, GCHART procedure 348
VER option, LIBNAME statement 7
  -VERBOSE initialization option 448
  %VERIFY macro function 406, 417
  versions, macro 427–430
VFORMAT function 149
VFORMATDX function 149
VFORMATN function 149
VFORMATNX function 149
VFORMATWX function 149
VFORMATWX function 149
VFORMATX function 149, 154
View Columns tool 6
VIEWTABLE command 451, 464
VIEWTABLE window (Display Manager)
  about 6, 200
  closing 452
  SASHELP views and 421
  showing column names in 450–451
VINARRAY function 149
VINARRAYX function 149
VINFORMAT function 149
VINFORMATD function 149
VINFORMATDX function 149
VINFORMATN function 149
VINFORMATNX function 149
VINFORMATW function 149
VINFORMATWX function 149
VINFORMATWX function 149
VLABEL function 149
VLABELX function 149
VLENGTH function 149
VLENGTHX function 150
VNAME function
  about 150, 183
  additional information 163
  usage example 153
VNAMEX function 150
VVALUE function 150
VVALUEX function 150, 153
~W sequence code 342–343
WAITFOR statement 479
warnings, customizing 474
WAYS option, OUTPUT statement 254
WAYS statement
  MEANS procedure 249–250
  SUMMARY procedure 249–250
WEDIT command 452
WHERE= data set option
  colon operator and 47
  creating WHERE clause 415–417
  in DATA steps 82–83
  SORT procedure and 191
WHERE statement
  about 82–83
  BY-group processing and 86–88
  checking date strings 53
  colon comparison operator modifier in 47
  compound inequalities and 49
  creating 415–417
  data set options and 45
  MIN and MAX operators 50–51
  negative values and 51
  operators supported 83–86
  PRINT procedure 351
  reordering variables on PDV and 201
WHICHN function 49, 162–163, 183
Whitlock, Ian 95, 176, 419, 427
WIDTH= option, SYMBOL statement 304
Windows fonts 319–320
WITHDEC. format 373
WINetGetConnectionA routine 470–472
%WORDCOUNT macro function 418–419
WORDDATE18. format 434
workbooks
See Excel sheets and workbooks
WORK.FORMATS catalog 393–394
WORK.SASMACR catalog 427–430
WRITE data set option 41
writing data
in delimited files 28–32
in e-mails 467–468
reports to Excel tables 332–336
to XML 33–35
with data access engines 5
writing macro functions 417–419
WRITFSAVE option, DM statement 440
X
X statement 79, 477–479
Xie, Liang 380
XMIN system option 478
XML (Extensible Markup Language)
EXCELPX tagset and 332
MARKUP destination 34
reading and writing to 33
XML engine 33–35
XML destination 33
XML engine 33–35
XMLFILEREF= option, LIBNAME statement 34
~xn sequence code 342–343
XPIXELS graphics option 321
XSYNC system option 479
XWAIT system option 478
~xz sequence code 342–343
Y
YEAR function 48, 116, 157
YESNO. format 395
YMDTIME. format 373
YPIXELS graphics option 321
YRDIF function 116–117
YYQ. format 387, 479–480
Z
.z missing value 98
Zdeb, Mike 154, 481
Zender, Cynthia 258

Symbols and Numbers
* (asterisk) 202, 410
@ (at sign) 26, 340
-(hyphen) 438–441
/ (slash) 239
~ sequence code 342–343
" (quotation marks)
about 79–81
DSD option and 21
macro language and 434–435, 475
# (pound sign) 350–351, 430–431
$ (dollar sign) 6, 386
% (percent sign) 84–85, 434–435
& (ampersand) 19–20, 434–435
& format modifier 18
() (parentheses) 119

About the Author

This is Art Carpenter’s fifth book and his publications list includes numerous papers and posters presented at SAS Global Forum, SUGI, and other user group conferences. Art is a SAS Silver Circle member and has been using SAS® since the mid 1970’s, and he has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Base Programmer for SAS 9, SAS Certified Clinical Trials Programmer Using SAS 9 and a SAS Certified Advanced Programmer for SAS 9. Through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

Author Contact
Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
http://www.caloxy.com
http://support.sas.com/publishing/authors/carpenter.html
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.