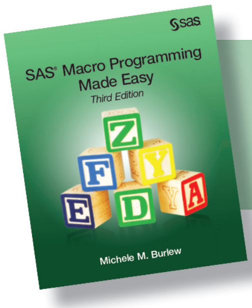
sas

SAS® Macro Programming Made Easy

Third Edition

Michele M. Burlew



From *SAS® Macro Programming Made Easy, Third Edition.*
Full book available for purchase here.

Contents

About This Book ix
About The Author xiii
Acknowledgments xv

Part 1: Understanding the Concepts and Features of Macro Programming 1

Chapter 1 Introduction 3
What Is the SAS Macro Facility? 4
What Are the Advantages of the SAS Macro Facility? 5
Where Can the SAS Macro Facility Be Used? 11
Examples of the SAS Macro Facility 12
Chapter 2 Mechanics of Macro Processing 21
Introduction 21
The Vocabulary of SAS Processing 21
SAS Processing without Macro Activity 22
Understanding Tokens 23
Tokenizing a SAS Program 24
Comparing Macro Language Processing and SAS Language Processing 26
Processing a SAS Program That Contains Macro Language 27
Chapter 3 Macro Variables 37
Introduction 37
Basic Concepts of Macro Variables 38
Referencing Macro Variables 39
Understanding Macro Variable Resolution and the Use of Single and Double Quotation Marks 41

Displaying Macro Variable Values 42
Using the %PUT Statement 43
Displaying Macro Variable Values As They Resolve by Enabling the SYMBOLGEN System Option 46
Understanding Automatic Macro Variables 48
Understanding User-Defined Macro Variables 51
Creating Macro Variables with the %LET Statement 52
Combining Macro Variables with Text 55
Placing Text before a Macro Variable Reference 55
Placing Text after a Macro Variable Reference 57
Concatenating Permanent SAS Data Set Names and Catalog Names with Macro Variables 58
Referencing Macro Variables Indirectly 60

Chapter 4 Macro Programs 67

Introduction 67
Creating Macro Programs 68
Executing a Macro Program 74
Displaying Notes about Macro Program Compilation in the SAS Log 75
Displaying Messages about Macro Program Processing in the SAS Log 77
Using MPRINT to Display the SAS Statements Submitted by a Macro Program 77
Using the MLOGIC Option to Trace Execution of a Macro Program 78
Passing Values to a Macro Program through Macro Parameters 79
Specifying Positional Parameters in Macro Programs 80
Specifying Keyword Parameters in Macro Programs 82
Specifying Mixed Parameter Lists in Macro Programs 86
Defining a Macro Program That Can Accept a Varying Number of Parameter Values . 89

Chapter 5 Understanding Macro Symbol Tables and the Processing of Macro Programs 95

Understanding Macro Symbol Tables 95
Understanding the Global Macro Symbol Table 97
Understanding Local Macro Symbol Tables 101
Working with Global Macro Variables and Local Macro Variables 108
Defining the Domain of a Macro Variable by Using the %GLOBAL or %LOCAL Macro Language Statements 110
Processing of Macro Programs 114

How a Macro Program Is Compiled ... 114
How a Macro Program Executes ... 120
Chapter 6 Macro Language Functions ... 129
Introduction ... 129
Macro Character Functions ... 130
Macro Evaluation Functions ... 133
Macro Quoting Functions ... 135
Macro Variable Attribute Functions ... 137
Other Macro Functions ... 141
SAS Supplied Autocall Macro Programs Used Like Functions ... 147
Chapter 7 Macro Expressions and Macro Programming Statements ... 151
Introduction ... 151
Macro Language Statements ... 151
Constructing Macro Expressions ... 154
Understanding Arithmetic Expressions ... 155
Understanding Logical Expressions ... 156
Understanding the IN Operator As Used in Macro Language Statements ... 157
Conditional Processing with the Macro Language ... 158
Iterative Processing with the Macro Language ... 167
Writing Iterative %DO Loops in the Macro Language ... 167
Conditional Iteration with %DO %UNTIL ... 170
Conditional Iteration with %DO %WHILE ... 172
Branching in Macro Processing ... 174
Chapter 8 Masking Special Characters and Mnemonic Operators ... 177
Introduction ... 177
Why Are Quoting Functions Called Quoting Functions? ... 178
Illustrating the Need for Macro Quoting Functions ... 179
Describing the Commonly Used Macro Quoting Functions ... 180
Understanding How Macro Quoting Functions Work ... 181
Applying Macro Quoting Functions ... 182
Specifying Macro Program Parameters That Contain Special Characters or Mnemonic Operators ... 189
Unmasking Text and the %UNQUOTE Function ... 198
Using Quoting Versions of Macro Character Functions and Autocall Macro Programs ... 198

Chapter 9 Interfaces to the Macro Facility 201

Introduction 201

Understanding DATA Step Interfaces to the Macro Facility 201

  Understanding the SYMGET Function 202

  Understanding the SYMPUT and SYMPUTX Call Routines 209

  Understanding the CALL EXECUTE Routine 216

  Understanding the RESOLVE Function 227

Using Macro Facility Features in PROC SQL 231

  Creating and Updating Macro Variables with PROC SQL 232

  Using the Macro Variables Created by PROC SQL 238

  Displaying Macro Option Settings with PROC SQL and Dictionary Tables 241

Part 2: Applying Your Knowledge of Macro Programming 245

Chapter 10 Storing and Reusing Macro Programs 247

Introduction 247

Saving Macro Programs with the Autocall Facility 248

  Creating an Autocall Library 248

  Making Autocall Libraries Available to Your Programs 250

  Maintaining Access to the Autocall Macro Programs That Ship with SAS 251

  Using the Autocall Facility under UNIX and z/OS Systems 252

Saving Macro Programs with the Stored Compiled Macro Facility 254

  Setting SAS Options to Create Stored Compiled Macro Programs 254

  Creating Stored Compiled Macro Programs 255

  Saving and Retrieving the Source Code of a Stored Compiled Macro Program 257

  Encrypting a Stored Compiled Macro Program 258

Resolving Macro Program References When Using the Autocall Facility and the Stored Compiled Macro Facility 258

Chapter 11 Building a Library of Utilities 261

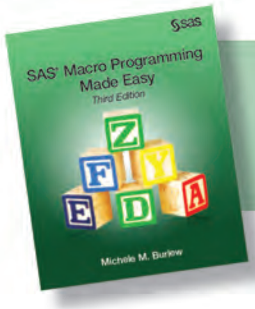
Introduction 261

Writing a Macro Program to Behave Like a Function 261

Programming Routine Tasks 266

Chapter 12 Debugging Macro Programming and Adding Error Checking to Macro Programs 273
Introduction 273
Understanding the Types of Errors That Can Occur in Macro Programming 273
Minimizing Errors in Developing SAS Programs That Contain Macro Language 274
Categorizing and Checking for Common Problems in Macro Programming 275
Understanding the Tools That Can Debug Macro Programming 278
    Using SAS System Options to Debug Macro Programming 278
    Using Macro Language Statements to Debug Macro Programming 279
    Using Macro Functions to Debug Macro Programming 280
    Using Automatic Macro Variables to Debug Macro Programming 280
Examples of Solving Errors in Macro Programming 281
Improving Your Macro Programming by Including Error Checking 298
Chapter 13 A Stepwise Method for Writing Macro Programs 307
Introduction 307
Building a Macro Program in Four Steps 307
Applying the Four Steps to an Example 308
    Step 1: Write, test, and debug the SAS program(s) that you want the macro program to build 309
    Step 2: Remove hard-coded programming constants from the program(s) in Step 1 and replace these constants with macro variables 317
    Step 3: Create macro program(s) from the program(s) in Step 2 321
    Step 4: Refine and generalize the macro program(s) in Step 3 by adding macro language statements like %IF-%THEN and %DO groups 325
    Executing the REPORT Macro Program 328
    Enhancing the Macro Program REPORT 338
Part 3 Appendixes 339
Appendix A Sample Data Set 341
Appendix B Reference to Examples in This Book 343
Index 355

From *SAS® Macro Programming Made Easy, Third Edition* by Michele M. Burlew. Copyright © 2014, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *SAS® Macro Programming Made Easy, Third Edition.*
Full book available for purchase here.

Chapter 6 Macro Language Functions

Introduction **129**
Macro Character Functions **130**
Macro Evaluation Functions **133**
Macro Quoting Functions **135**
Macro Variable Attribute Functions **137**
Other Macro Functions **141**
Using the %SYSFUNC and %QSYSFUNC Macro Functions 141
SAS Supplied Autocall Macro Programs Used Like Functions **147**

Introduction

The preceding chapters describe the basic structures of the macro programming language and the mechanics involved in processing macro language. This chapter describes the functions that are available in the macro programming language.

Macro functions greatly extend the use of macro variables and macro programming. Macro functions can be used in open code and in macro programs. The arguments of a macro function can be text strings, macro variables, macro functions, and macro program calls. The *result* of a macro function is always text. This result can be assigned to a macro variable. A macro function can also be inserted directly into your SAS statements to build SAS statements.

Most macro functions have SAS language counterparts. If you know how to write DATA step programs, you already have a familiarity with the style and structure of many of the macro functions.

Some of the tasks you can do with macro functions include:

- extracting substrings of macro variables
- searching for a string of characters in a macro variable

- temporarily converting macro values to numeric so that you can use the macro variables in calculations
- using SAS language functions and functions created with PROC FCMP in your macro language statements
- allowing semicolons to be treated as a text value rather than as a symbol to terminate a statement

This chapter classifies the macro functions into five categories: character, evaluation, quoting, macro variable attribute, and other. These categories and some of the functions in each category are briefly described below.

Additionally, SAS ships a library of autocall macro programs with its software, which may or may not be installed at your site. Autocall macro programs are uncompiled source code and text stored as entries in SAS libraries. This set of autocall macro programs can be used like macro language functions. This SAS supplied autocall macro program library is described at the end of this chapter. Also see Chapter 10 for more discussion on the application of autocall macro programs.

Macro Character Functions

Macro character functions operate on strings of text or on macro variables. They can modify their arguments, return information about an argument, or return text values. Several of the character functions you might be familiar with in the SAS language have macro language counterparts. Table 6.1 lists the macro character functions.

Macro functions %SCAN, %SUBSTR, and %UPCASE each have a version that should be used instead if the result of the macro function might contain a special character or mnemonic operator. The names of these macro functions are %QSCAN, %QSUBSTR, and %QUPCASE.

Table 6.1 Macro character functions

Function	**Action**
`%INDEX(`*`source, string`*`)`	returns the position in *`source`* of the first character of *`string.`*
`%LENGTH(`*`string|text expression`*`)`	returns the length of *`string`* or the length of the results of the resolution of *`text expression.`*

`%SCAN(`*`argument,`*
      `n <,charlist<,modifiers>>``)`
	searches for the *`n`*th word in *`argument`*, where *`argument`* can be text or an expression and the words in *`argument`* are separated by *`charlist`*. Specify *`modifiers`* to modify how %SCAN determines word boundaries in

	`argument`. Use %QSCAN when you need to mask special characters or mnemonic operators in the result.
%SUBSTR(*argument,position<,length>)*	extracts a substring of *length* characters from *argument* starting at *position.* Use %QSUBSTR when you need to mask special characters or mnemonic operators in the result.
%UPCASE(*string|text expression)*	converts *character string* or *text expression* to uppercase. Use %QUPCASE when you need to mask special characters or mnemonic operators in the result.

Example 6.1: Using %SUBSTR to Extract Text from a Macro Variable Value

Example 6.1 shows how the %SUBSTR function extracts text from strings of characters. The WHERE statement in the PROC MEANS step selects observations from the first day of the current month through the day the program was run. The current month is determined by extracting that information from the value of automatic macro variable SYSDATE.

```
proc means data=books.ytdsales;
  title "Sales for 01%substr(&sysdate,3,3) through &sysdate9";
  where "01%substr(&sysdate,3)"d le datesold le "&sysdate"d;
  class section;
  var saleprice;
run;
```

After resolution of the macro variable references, the PROC MEANS step looks as follows when submitted on September 15, 2014.

```
proc means data=books.ytdsales;
  title "Sales for 01SEP through 15SEP2014";
  where "01SEP13"d le datesold le "15SEP14"d;
  class section;
  var saleprice;
run;
```

Example 6.2: Using %SCAN to Extract the Nth Item from a Macro Variable Value

The %SCAN macro character function in Example 6.2 extracts a specific word from a string of words that are separated with blanks. The code specifies that %SCAN, through the value of REPMONTH, extract the third word from macro variable MONTHS.

One of the default delimiters for %SCAN is a blank. Therefore, Example 6.2 does not specify the optional third argument to %SCAN.

Under ASCII systems, the other default delimiters for %SCAN are:

```
! $ % & ( ) * + , - . / ; < ^¦
```

Under EBCDIC systems, the other default delimiters for %SCAN are:

```
! $ % & ( ) * + , - . / ; < ¬ | ¢¦
```

While not used in this simple example, you can add modifiers to %SCAN to alter how %SCAN determines boundaries between items in your argument string. For example, if you add the B modifier, the macro processor scans your argument string from right to left instead of the default direction of left to right.

```
%let months=January February March April May June;
%let repmonth=3;

proc print data=books.ytdsales;
  title "Sales Report for %scan(&months,&repmonth)";
  where month(datesold)=&repmonth;
  var booktitle author saleprice;
run;
```

After resolution of the macro variable references, the PROC PRINT step becomes:

```
proc print data=books.ytdsales;
  title "Sales Report for March";
  where month(datesold)=3;
  var booktitle author saleprice;
run;
```

Example 6.3: Using %UPCASE to Convert a Macro Variable Value to Uppercase

Macro program LISTTEXT in Example 6.3 lists all the titles sold that contain a specific text string. The text string is passed to the macro program through the parameter KEYTEXT. This text string might be in different forms in the title: lowercase, uppercase, or mixed case. Because of this, both the macro variable's value and the value of the data set variable BOOKTITLE are converted to uppercase. This increases the likelihood of matches when the two are compared.

```
%macro listtext(keytext);
  %let keytext=%upcase(&keytext);
  proc print data=books.ytdsales;
  title "Book Titles Sold Containing Text String &keytext";
    where upcase(booktitle) contains "&keytext";
    var booktitle author saleprice;
  run;
%mend;

%listtext(web)
```

When the macro program executes, the TITLE statement resolves to

```
   Book Titles Sold Containing Text String WEB
```

The WHERE statement at execution resolves to

```
   where upcase(booktitle) contains "WEB";
```

Macro Evaluation Functions

The two macro evaluation functions, %EVAL and %SYSEVALF, evaluate arithmetic expressions and logical expressions. These expressions are comprised of operators and operands that the macro processor evaluates to produce a result. The arguments to one of these macro evaluation functions are temporarily converted to numbers so that a calculation (arithmetic or logical) can be completed. The macro evaluation function converts the result that it returns to text.

Arithmetic expressions use arithmetic operators such as plus signs and minus signs. Logical expressions use logical operators such as greater than signs and equal signs.

The %EVAL function evaluates expressions using integer arithmetic. The %SYSEVALF function evaluates expressions using floating point arithmetic. Macro expressions are constructed with the same arithmetic and comparison operators found in the SAS language. A section in Chapter 7 discusses in more detail how to construct macro expressions.

The syntax of the %EVAL function is

```
   %EVAL(arithmetic expression|logical expression)
```

The syntax of the %SYSEVALF function is

```
   %SYSEVALF(arithmetic expression|logical expression
       <,conversion-type>)
```

By default, the result of the %SYSEVALF function is left as a number which the macro processor converts back to text. Otherwise, you can request %SYSEVALF to convert the result to a different format, as shown in Table 6.2. When you want to use one of these four conversion types, specify it as the second argument to %SYSEVALF.

Table 6.2 Conversion types that can be specified on the %SYSEVALF function

Conversion Type	Result that is returned by %SYSEVALF
BOOLEAN	0 if the result of the expression is 0 or null 1 if the result is any other value

	(The 0 and 1 are treated as text.)
`CEIL`	text that represents the smallest integer that is greater than or equal to the result of the expression
`FLOOR`	text that represents the largest integer that is less than or equal to the result of the expression
`INTEGER`	text that represents the integer portion of the expression's result

For more discussion of the usage of %SYSEVALF and %EVAL, see Example 6.8 later in this chapter.

The %EVAL function does *integer arithmetic*. Therefore, this function treats numbers with decimal points as text. The %EVAL function generates an error when there are characters in the arguments that are supplied to %EVAL, as demonstrated by the second example in Table 6.3.

The statements in Table 6.3 show examples of the %EVAL and %SYSEVALF functions. The %PUT statements were submitted, and the results were written to the SAS log.

Table 6.3 Examples of %EVAL and %SYSEVALF evaluation functions

%PUT Statement	Results in SAS log
`%put %eval(33 + 44);`	`77`
`%put %eval(33.2 + 44.1);`	`ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: 33.2 + 44.1`
`%put %sysevalf(33.2 + 44.1);`	`77.3`
`%put %sysevalf(33.2 + 44.1,integer);`	`77`
`%let a=3;` `%let b=10;`	
`%put %eval(&b/&a);`	`3`
`%put %sysevalf(&b/&a);`	`3.3333333333333`
`%put %sysevalf(&b/&a,ceil);`	`4`
`%put %sysevalf(&b/&a,floor);`	`3`
`%put %sysevalf(&b/&a,boolean);`	`1`

%PUT Statement	Results in SAS log

`%let missvalue=.;`
`%put %sysevalf(&b-&missvalue, boolean);`
	`NOTE: Missing values were generated as a result of performing an operation on missing values during %SYSEVALF expression evaluation.`
	`0`

Macro Quoting Functions

Macro quoting functions mask special characters and mnemonic operators in your macro language statements so that the macro processor does not interpret them. The macro processor instead treats these items simply as text.

For example, you might want to assign a value to a macro variable that contains a character that the macro processor interprets as a macro trigger. The macro processor considers ampersands and percent signs followed by text as macro triggers. You must use a macro quoting function to tell the macro processor to ignore the special meaning of the ampersands and percent signs and instead treat them as text.

Consider what happens if you assign the name of the publisher, Doe&Lee Ltd., to a macro variable:

```
%let publisher=Doe&Lee Ltd.;
```

If you have not already defined a macro variable named LEE in your SAS session, you will see the following message displayed in your SAS log:

```
WARNING: Apparent symbolic reference LEE not resolved.
```

(If you had already defined a macro variable named LEE in your SAS session, you would not see the warning. Instead, the macro processor would resolve the reference to &LEE with the value assigned to the macro variable LEE.)

To prevent the macro processor from interpreting the ampersand as a macro trigger in the value being assigned to PUBLISHER, you must mask the value that you assign to the macro variable PUBLISHER. The macro quoting function %NRSTR correctly masks &LEE from view by the macro processor when it compiles the instruction. Therefore, when you apply %NRSTR to the text string, the macro processor ignores the ampersand as a macro trigger; does not attempt to resolve the value of the macro variable &LEE; and considers this use of the ampersand simply as text.

```
%let publisher=%nrstr(Doe&Lee Ltd.);
```

As it steps through its tasks in compiling and executing the %LET statement, the macro processor defines a global macro variable, PUBLISHER, and assigns the text `Doe&Lee Ltd.` to PUBLISHER.

The macro quoting functions can be grouped into two types based upon when they act: compilation and execution.

Table 6.4 lists the macro quoting functions. Chapter 8 discusses the topic of masking characters in macro programming more thoroughly, and it includes several examples that illustrate the concepts on how and when to apply the macro quoting functions.

Table 6.4 Macro quoting functions

Function	**Action**
`%BQUOTE(`*`character-string`*` | `*`text expression`*`)`	Mask special characters and mnemonic operators in a character string or the value of resolved text expression at macro execution. Compared to %QUOTE, %BQUOTE does not require that unmatched quotation marks or unmatched parentheses be marked with a preceding percent sign (%).
`%NRBQUOTE(`*`character-string`*` | `*`text expression`*`)`	Does the same as %BQUOTE and additionally masks ampersands (&) and percent signs (%). However, SAS recommends that you use %SUPERQ instead of %NRBQUOTE.
`%QUOTE(`*`character-string`*` | `*`text expression`*`)`	Mask special characters and mnemonic operators in a character string or the value of resolved text expression at macro execution. Compared to %BQUOTE, %QUOTE requires that unmatched quotation marks and unmatched parentheses be marked with a preceding percent sign (%).
`%NRQUOTE(`*`character-string`*` | `*`text expression`*`)`	Does the same as %QUOTE and additionally masks ampersands (&) and percent signs (%).However, SAS recommends that you use %SUPERQ instead of %NRQUOTE.
`%STR(`*`character-string)`*	Mask special characters and mnemonic operators in constant text at macro compilation.
`%NRSTR(`*`character-string)`*	Does the same as %STR and additionally masks ampersands (&) and percent signs (%).

Function	**Action**
`%SUPERQ(`*`macro-variable-name)`*	Masks all special characters including ampersands (&), percent signs (%), and mnemonic operators at macro execution and prevents further resolution of the value. Returns the value of a macro variable without attempting to resolve any macro program or macro variable references in the value.
`%UNQUOTE(`*`character-string`* `|` *`text expression`*`)`	Unmasks all special characters and mnemonic operators in a value at macro execution.

Macro Variable Attribute Functions

Table 6.5 lists the three macro variable attribute functions that supply information about the existence and the domain (global vs. local) of macro variables. These functions can be especially useful when debugging problems with macro variable resolution.

Table 6.5 Macro variable attribute functions

Function	**Action**
`%SYMEXIST(`*`macro-variable-name`*`)`	Determines whether the named macro variable exists. The search starts with the most local symbol table, and the search proceeds up the hierarchy through other local symbol tables, ending the search at the global symbol table. If the macro variable exists, %SYMEXIST returns a value of 1; otherwise, it returns a 0.
`%SYMGLOBL(`*`macro-variable-name`*`)`	Determines whether the named macro variable is found in the global symbol table. If the macro variable exists in the global symbol table, %SYMGLOBL returns a value of 1; otherwise, it returns a 0.
`%SYMLOCAL(`*`macro-variable-name`*`)`	Determines whether the named macro variable is found in a local symbol table. The search starts with the most local symbol table, and the search proceeds up the hierarchy through other local symbol tables, ending the search at the local symbol table highest up in the hierarchy. If the macro variable exists in a local symbol table, %SYMLOCAL returns a value of 1; otherwise, it returns a 0.

Example 6.4: Using Macro Variable Attribute Functions to Determine Domain and Existence of Macro Variables

Chapter 5 discusses domains of macro variables. A macro variable can exist in either the global or local macro symbol table. You can successfully reference a macro variable stored in the global symbol table throughout your SAS session including within macro programs. There is only one global symbol table.

A local macro symbol table is created by executing a macro program that contains macro variables. If the macro variables do not already exist in the global table, macro variables defined in the macro program are stored in the local macro symbol table associated with the macro program. These local macro variables can be referenced only from within the macro program. The macro processor deletes a local macro symbol table when the macro program associated with the table ends. You can have more than one local macro symbol table at a time if one macro program calls another.

Example 5.4 in Chapter 5, which demonstrates domains of macro variables, is modified below in Example 6.4 to include the three functions described in Table 6.5 and to illustrate their use. This program introduces a new statement, %SYMDEL, which deletes macro variables from the global symbol table.

```
%* For example purposes only, ensure these two macro
   variables do not exist in the global symbol table;
%symdel glbsubset subset;

%macro makeds(subset);
  %global glbsubset;
  %let glbsubset=&subset;

   %* What is domain of SUBSET and GLBSUBSET inside MAKEDS?;
   %put ******** Inside macro program;
   %put Is SUBSET a local macro variable(0=No/1=Yes):
%symlocal(subset);
   %put Is SUBSET a global macro variable(0=No/1=Yes):
%symglobl(subset);
   %put Is GLBSUBSET a local macro variable(0=No/1=Yes):
%symlocal(glbsubset);
   %put Is GLBSUBSET a global macro variable(0=No/1=Yes):
%symglobl(glbsubset);
   %put ********;

  data temp;
    set books.ytdsales(where=(section="&subset"));
    attrib qtrsold label='Quarter of Sale';
    qtrsold=qtr(datesold);
  run;
%mend makeds;

%makeds(Software)
```

```
%* Are SUBSET and GLBSUBSET in global symbol table?;
%put Does SUBSET exist (0=No/1=Yes): %symexist(subset);
%put Is SUBSET a global macro variable(0=No/1=Yes): %symglobl(subset);
%put Is GLBSUBSET a global macro variable(0=No/1=Yes):
%symglobl(glbsubset);

proc tabulate data=temp;
  title "Book Sales Report Produced &sysdate9";
  class qtrsold;
  var saleprice listprice;
  tables qtrsold all,
         (saleprice listprice)*(n*f=6. sum*f=dollar12.2) /
         box="Section: &glbsubset";
  keylabel all='** Total **';
run;
```

The following SAS log for the program shows how the functions %SYMLOCAL, %SYMGLOBL, and %SYMEXIST resolve in this example.

```
288  %* For example purposes only, ensure these two macro
289     variables do not exist in the global symbol table;
290
291  %symdel glbsubset subset;
WARNING: Attempt to delete macro variable GLBSUBSET failed.
         Variable not found.
WARNING: Attempt to delete macro variable SUBSET failed.
         Variable not found.
292
293  %macro makeds(subset);
294    %global glbsubset;
295    %let glbsubset=&subset;
296
297     %* What is domain of SUBSET and GLBSUBSET inside MAKEDS?;
298     %put ******** Inside macro program;
299     %put Is SUBSET a local macro variable(0=No/1=Yes):
299! %symlocal(subset);
300     %put Is SUBSET a global macro variable(0=No/1=Yes):
300! %symglobl(subset);
301     %put Is GLBSUBSET a local macro variable(0=No/1=Yes):
301! %symlocal(glbsubset);
302     %put Is GLBSUBSET a global macro variable(0=No/1=Yes):
302! %symglobl(glbsubset);
303     %put ********;
304
305    data temp;
306      set books.ytdsales(where=(section="&subset"));
307      attrib qtrsold label='Quarter of Sale';
308      qtrsold=qtr(datesold);
309    run;
310  %mend makeds;
```

```
311
312  %makeds(Software)
******** Inside macro program
Is SUBSET a local macro variable(0=No/1=Yes): 1
Is SUBSET a global macro variable(0=No/1=Yes): 0
Is GLBSUBSET a local macro variable(0=No/1=Yes): 0
Is GLBSUBSET a global macro variable(0=No/1=Yes): 1
********

NOTE: There were 857 observations read from the data set
      BOOKS.YTDSALES.
      WHERE section='Software';
NOTE: The data set WORK.TEMP has 857 observations and 11
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds

313
314  %* Are SUBSET and GLBSUBSET in global symbol table?;
315  %put Does SUBSET exist (0=No/1=Yes):%symexist(subset);
Does SUBSET exist (0=No/1=Yes): 0
316  %put Is SUBSET a global macro variable(0=No/1=Yes):
316! %symglobl(subset);
Is SUBSET a global macro variable(0=No/1=Yes): 0
317  %put Is GLBSUBSET a global macro variable(0=No/1=Yes):
317! %symglobl(glbsubset);
Is GLBSUBSET a global macro variable(0=No/1=Yes): 1
318
319  proc tabulate data=temp;
320    title "Book Sales Report Produced &sysdate9";
321    class qtrsold;
322    var saleprice listprice;
323    tables qtrsold all,
324           (saleprice listprice)*(n*f=6. sum*f=dollar12.2) /
325           box="Section: &glbsubset";
326    keylabel all='** Total **';
327  run;

NOTE: There were 857 observations read from the data set
      WORK.TEMP.
NOTE: PROCEDURE TABULATE used (Total process time):
      real time           0.12 seconds
      cpu time            0.03 seconds
```

Other Macro Functions

The four macro functions described in this section (Table 6.6) do not fit into any of the four categories of macro functions described so far. These four functions do one of the following:

- apply SAS language functions to macro variables or text
- obtain information from the rest of SAS or the operating system in which the SAS session is running

The %SYSFUNC and %QSYSFUNC functions are especially useful in extending the use of the macro facility. These functions allow you to apply SAS language and user-written functions to your macro programming applications. Several examples of %SYSFUNC follow. Chapter 8 presents the topic of masking special characters and mnemonic operators, and it also includes an example that applies %QSYSFUNC. Macro function %QSYSFUNC does the same as %SYSFUNC, and it masks special characters and mnemonic operators in the result.

Table 6.6 Other macro functions

Function	**Action**
`%SYSFUNC(`*`function(argument(s))`* `<,`*`format`*`>)`	executes SAS *`function`* or user-written *`function`* and returns the results to the macro facility (see also macro statement %SYSCALL)
`%QSYSFUNC(`*`function(argument(s))`* `<,`*`format`*`>)`	does the same as %SYSFUNC and it also masks special characters and mnemonic operators in the result
`%SYSGET(`*`host-environment-variable`*`)`	returns the value of *`host-environment-variable`* to the macro facility
`%SYSPROD(`*`SAS-product`*`)`	returns a code to indicate whether *`SAS-product`* is licensed at the site where SAS is currently running

Using the %SYSFUNC and %QSYSFUNC Macro Functions

The functions %SYSFUNC and %QSYSFUNC apply SAS programming language functions to text and macro variables in your macro programming. Providing access to the many SAS language functions in your macro programming applications, %SYSFUNC and %QSYSFUNC greatly extend the power of your macro programming.

Since these two functions are macro language functions and the macro facility is a text-handling language, the arguments to the SAS programming language function are not enclosed in quotation marks; it is understood that all arguments are text. Also, the values returned through the use of these two functions are considered text.

Functions cannot be nested within one call to %SYSFUNC and %QSYSFUNC. Each function must have its own %SYSFUNC or %QSYSFUNC call, and these %SYSFUNC and %QSYSFUNC calls can be nested.

Example 6.5: Using %SYSFUNC to Format a Date in the TITLE Statement

The TITLE statement in Example 6.5 shows how the elements of a date can be formatted using %SYSFUNC and the DATE SAS language function.

```
title
  "Sales for %sysfunc(date(),monname.) %sysfunc(date(),year.)";
```

On January 30, 2014, the title statement would resolve to

```
Sales for January 2014
```

Example 6.6: Using %SYSFUNC to Execute a SAS Language Function and Assign the Result to a Macro Variable

Example 6.6 uses %SYSFUNC to access the SAS language function GETOPTION. The GETOPTION function displays the values of SAS options. The %SYSFUNC function invokes the GETOPTION function and returns the result to macro variable OPTVALUE. The %PUT statement lists the value assigned to OPTVALUE. The single parameter to GETOPTION is the name of the SAS option that should be checked.

```
%macro getopt(whatopt);
  %let optvalue=%sysfunc(getoption(&whatopt));
  %put Option &whatopt = &optvalue;
%mend getopt;

%getopt(number)
%getopt(orientation)
%getopt(date)
%getopt(symbolgen)
%getopt(compress)
```

The SAS log for Example 6.6 follows.

```
58   %getopt(number)
Option number = NUMBER
59   %getopt(orientation)
Option orientation = PORTRAIT
60   %getopt(date)
Option date = DATE
```

```
61   %getopt(symbolgen)
Option symbolgen = NOSYMBOLGEN
62   %getopt(compress)
Option compress = NO
```

Example 6.7: Using %SYSFUNC and the NOTNAME and NVALID SAS Language Functions to Determine If a Value Is a Valid SAS Variable Name

Since the macro language generates SAS code for you, a common task that macro programmers have is to construct SAS items such as variable names and format names. In doing so, it might be important to check the value that will be used to name the item to make sure that it does not contain any invalid characters or is too long.

Macro program CHECKVARNAME in Example 6.7 checks a SAS macro variable value to see if it can be used as a variable name. It uses both the NOTNAME and the NVALID SAS functions. The NOTNAME function finds the first position in the value that is an invalid character in naming a variable. The NVALID function determines if the value can be used as a variable name.

The second argument to NVALID in this example is V7. This argument requires three conditions to be true if it is to be determined to be valid:

- The value must start with a letter or underscore.
- All subsequent characters must be letters, underscores, or digits.
- Its length must be no greater than 32 characters.

The parameter passed to CHECKVARNAME is the prospective variable name that should be examined. The %SYSFUNC macro function is used in conjunction with the NOTNAME SAS language function and again with the NVALID SAS language function. The macro program writes messages to the SAS log about whether the value can be used as a variable name.

Example 6.7 calls CHECKVARNAME four times. The parameter values specified for the first two calls to CHECKVARNAME are valid SAS names. The parameter values specified in the third and fourth calls to CHECKVARNAME are not valid SAS names. The space in the parameter in the third call to CHECKVARNAME is invalid. The length of the parameter in the fourth call, as well as the exclamation point in the last position, makes the value invalid as a SAS name.

```
%macro checkvarname(value);
  %let position=%sysfunc(notname(&value));
  %put **** Invalid character in position: &position (0 means &value
is okay);
  %let valid=%sysfunc(nvalid(&value,v7));
  %put
     **** Can &value be a variable name(0=No, 1=Yes)? &valid;
  %put;
  %put;
%mend checkvarname;
```

```
%checkvarname(valid_name)
%checkvarname( valid_name)
%checkvarname(invalid name)
%checkvarname(book_sales_results_for_past_five_years!)
```

The four calls to macro program CHECKVARNAME produce the following SAS log.

```
235  %checkvarname(valid_name)
**** Invalid character in position: 0 (0 means valid_name is okay)
**** Can valid_name be a variable name(0=No, 1=Yes)? 1

236  %checkvarname( valid_name)
**** Invalid character in position: 0 (0 means valid_name is okay)
**** Can valid_name be a variable name(0=No, 1=Yes)? 1

237  %checkvarname(invalid name)
**** Invalid character in position: 8 (0 means invalid name is okay)
**** Can invalid name be a variable name(0=No, 1=Yes)? 0

238  %checkvarname(book_sales_results_for_past_five_years!)
**** Invalid character in position: 39 (0 means
book_sales_results_for_past_five_years! is okay)
**** Can book_sales_results_for_past_five_years! be a variable
name(0=No, 1=Yes)? 0
```

Example 6.8: Using %SYSFUNC to Apply a SAS Statistical Function to Macro Variable Values

This example uses %SYSFUNC to apply the SAS statistical function MEAN to four macro variable values and compute their mean. In addition to using a SAS language function in the macro programming environment, this example also illustrates several concepts of macro programming.

The values assigned to the four macro variables A, B, C, and D are treated as *text* values in the macro programming environment. However, note that the MEAN function interprets them as *numbers*. The %SYSEVALF function is not needed to temporarily convert the values to numbers in order to compute the mean. Note also that two periods follow &MEANSTAT in the %PUT statement. The first period terminates the macro variable reference. The second period appears in the text written to the SAS log.

```
%let a=1.5;
%let b=-2.0;
%let c=1.978;
%let d=-3.5;
%let meanstat=%sysfunc(mean(&a,&b,&c,&d));
%put ****** The mean of &a, &b, &c, and &d is &meanstat..;
```

After the above code is submitted, the following is written to the SAS log:

```
****** The mean of 1.5, -2.0, 1.978, and -3.5 is -0.5055.
```

A section earlier in this chapter describes the necessity of using the %EVAL and %SYSEVALF functions when you need to temporarily convert macro variable values to numbers to perform calculations. In Example 6.8, if you wanted to compute the mean using only macro language statements, you would need to use the %SYSEVALF function. You could not use the %EVAL function because the values include decimal places. Also, %EVAL would return an integer result, which would be inaccurate. Code that includes %SYSEVALF follows.

```
%let a=1.5;
%let b=-2.0;
%let c=1.978;
%let d=-3.5;
%let meanstat=%sysevalf( (&a+&b+&c+&d)/4);
%put ****** The mean of &a, &b, &c, and &d is &meanstat..;
```

After submitting this code, the macro processor writes the same text statement to the SAS log as the one generated by the code that uses %SYSFUNC and MEAN in the first group of statements in this example:

```
****** The mean of 1.5, -2.0, 1.978, and -3.5 is -0.5055.
```

Example 6.9: Using the %SYSFUNC Function to Apply Several SAS Language Functions That Obtain and Display Information about a Data Set

Example 6.9 uses the %SYSFUNC macro function and several SAS language file functions to obtain the last date and time that a data set was updated and to insert that descriptive information in the title of a report. It also uses %SYSFUNC to format the date/time value.

The name of the data set is assigned to macro variable DSNAME. The data set specified by the value of DSNAME is opened with the SAS language OPEN function. Then, the SAS language ATTRN function obtains the last update information by specifying the argument MODTE. The results of the ATTRN function are stored in the macro variable LASTUPDATE. Finally, the SAS language CLOSE function closes the data set.

The value returned by ATTRN is the SAS internal date/time value, and this value is formatted for display in the title with the DATETIME format. The format is applied to the macro variable value stored in LASTUPDATE with %SYSFUNC and the PUTN SAS language function. The second argument to PUTN is the format name DATETIME.

Note that none of the arguments to the SAS file functions are enclosed in quotation marks. This is because the macro facility is a text-handling language and it treats all values as text. The SAS language functions are underlined in this example.

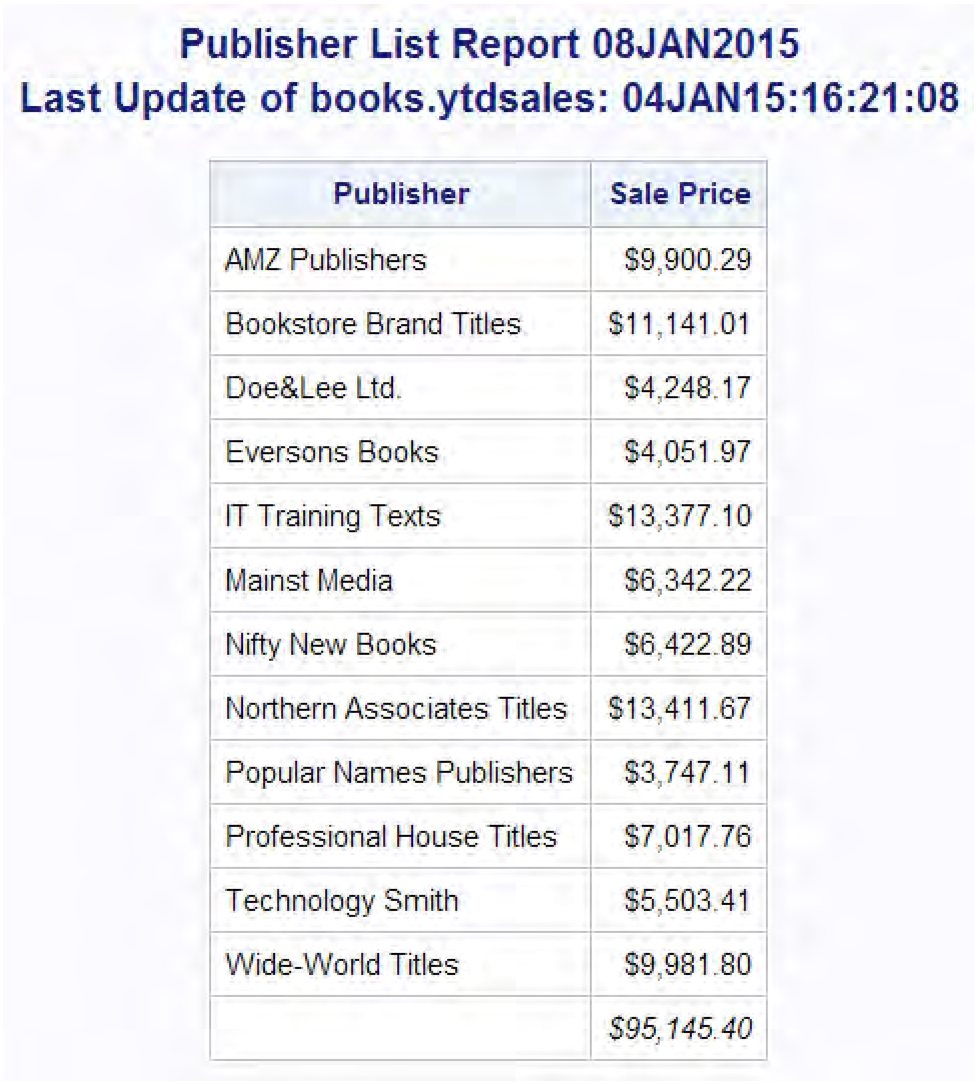
```
%let dsname=books.ytdsales;
%let dsid=%sysfunc(open(&dsname));
%let lastupdate=%sysfunc(attrn(&dsid,modte));
%let rc=%sysfunc(close(&dsid));

proc report data=books.ytdsales nowd;
  title "Publisher List Report &sysdate9";
  title2 "Last Update of &dsname:
%sysfunc(putn(&lastupdate,datetime.))";

  column publisher saleprice;
  define publisher / group;
  define saleprice / format=dollar11.2;
  rbreak after / summarize;
run;
```

Output 6.1 presents the output from Example 6.9.

Output 6.1 Output from Example 6.9



Publisher List Report 08JAN2015
Last Update of books.ytdsales: 04JAN15:16:21:08

Publisher	Sale Price
AMZ Publishers	$9,900.29
Bookstore Brand Titles	$11,141.01
Doe&Lee Ltd.	$4,248.17
Eversons Books	$4,051.97
IT Training Texts	$13,377.10
Mainst Media	$6,342.22
Nifty New Books	$6,422.89
Northern Associates Titles	$13,411.67
Popular Names Publishers	$3,747.11
Professional House Titles	$7,017.76
Technology Smith	$5,503.41
Wide-World Titles	$9,981.80
	$95,145.40

SAS Supplied Autocall Macro Programs Used Like Functions

Autocall macro programs are uncompiled source code and text stored as entries in SAS libraries. SAS has written several of these useful macro programs and ships them with SAS software. Not all SAS sites, however, install this autocall macro library, and some autocall macro programs can be site-specific as well.

These macro programs can be used like macro functions in your macro programming. Many of the functions perform actions comparable to their similarly named SAS language counterpart. For example, one autocall macro program is %LOWCASE. This autocall macro program converts alphabetic characters in its argument to lowercase. Similarly, you could use %SYSFUNC and the LOWCASE SAS language function to do the same action. Chapter 10 discusses how you can save your macro programs in your own autocall libraries.

Table 6.7 lists several of the autocall macro programs. Autocall macro programs %CMPRES, %LEFT, and %LOWCASE each have a version that you should use if the result might contain a special character or mnemonic operator. The names of those autocall macro programs are: %QCMPRES, %QLEFT, and %QLOWCASE.

Table 6.7 Selected SAS supplied autocall macro programs

Function	**Action**
`%CMPRES(`*`text | text expression`*`)`	Remove multiple, leading, and trailing blanks from the argument. Use %QCMPRES if the result might contain a special character or mnemonic operator.
`%DATATYP(`*`text | text expression`*`)`	Returns the data type (CHAR or NUMERIC) of a value.
`%LEFT(`*`text | text expression`*`)`	Aligns an argument to the left by removing leading blanks. Use %QLEFT if the result might contain a special character or mnemonic operator.
`%LOWCASE(`*`text | text expression`*`)`	Changes a value from uppercase characters to lowercase. Use %QLOWCASE if the result might contain a special character or mnemonic operator.
`%TRIM(`*`text | text expression`*`)`	Remove trailing blanks from the argument. Use %QTRIM if the result might contain a special character or mnemonic operator.
`%VERIFY(`*`source | excerpt`*`)`	Returns the position of the first character unique to an expression.

Example 6.10: Determining with %VERIFY and %UPCASE If a Value Is in a Defined Set of Characters

This example examines the value of a macro variable to see if its value is a valid response to a survey question. Macro program CHECKSURVEY in the first section of code has one parameter, RESPONSE. The value of RESPONSE is examined, and the result of the examination is printed in the SAS log with the %PUT statement. A response to a survey question in this example must be a digit from 1 to 5 or 9, or a letter from A to E or Z.

The example converts the survey response value to uppercase with the %UPCASE macro function. It then examines the value with the %VERIFY autocall macro program. The %VERIFY autocall macro program returns the position in a text value of the first character that is not in the list supplied as the second argument. In this example, assume that survey responses are single characters, and only single characters will be specified as parameters to CHECKSURVEY. Therefore, the call to %VERIFY in this example can return only one of two values: zero (0) for a valid response and one (1) for an invalid response. The returned value of 1 corresponds to the first and only character specified as the parameter to CHECKSURVEY.

Since the value of macro variable RESPONSE is converted to uppercase, only the uppercase letters of the alphabet are specified in the list of valid responses assigned to the macro variable VALIDRESPONSES. Note that the string of valid values is not enclosed in quotation marks. Since %VERIFY is a macro program, it treats all values as text and therefore you do not enclose the text in quotation marks as you would when processing text in SAS language statements.

```
%macro checksurvey(response);
  %let validresponses=123459ABCDEZ;
  %let result=%verify(%upcase(&response),&validresponses);
  %put ******* Response &response is valid/invalid (0=valid
1=invalid): &result;
%mend checksurvey;

%checksurvey(f)
%checksurvey(a)
%checksurvey(6)
```

After submitting the preceding three calls to macro program CHECKSURVEY, the following is written to the SAS log:

```
175  %checksurvey(f)
******* Response f is valid/invalid (0=valid 1=invalid): 1
176  %checksurvey(a)
******* Response a is valid/invalid (0=valid 1=invalid): 0
177  %checksurvey(6)
******* Response 6 is valid/invalid (0=valid 1=invalid): 1
```

The same information can be obtained by using the %SYSFUNC macro function in conjunction with the VERIFY and UPCASE SAS language functions. The following program uses %SYSFUNC, VERIFY, and UPCASE.

Note that two calls to %SYSFUNC are made, once for each of the two SAS language functions. As mentioned at the beginning of this section, you cannot nest multiple calls to SAS language functions within one call to %SYSFUNC, but you can nest multiple %SYSFUNC calls.

This example nests only one %SYSFUNC call. When you need to have multiple %SYSFUNC calls, it might be easier to step through the processing by specifying multiple %LET statements rather than trying to nest several calls on one %LET statement. Doing so can prevent frustrating debugging tasks as you figure out the proper positioning of all the parentheses, commas, and arguments.

The SAS language functions are underlined in this revised version of Example 6.1.

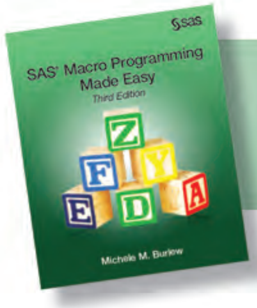
```
%macro checksurvey(response);
  %let validresponses=123459ABCDEZ;
  %let result=
%sysfunc(verify(%sysfunc(upcase(&response)),&validresponses));
  %put ******* Response &response is valid/invalid (0=valid
1=invalid): &result;
%mend checksurvey;

%checksurvey(f)
%checksurvey(a)
%checksurvey(6)
```

After submitting the above statements, the following is written to the SAS log:

```
193  %checksurvey(f)
******* Response f is valid/invalid (0=valid 1=invalid): 1
194  %checksurvey(a)
******* Response a is valid/invalid (0=valid 1=invalid): 0
195  %checksurvey(6)
******* Response 6 is valid/invalid (0=valid 1=invalid): 1
```

From *SAS® Macro Programming Made Easy, Third Edition* by Michele M. Burlew. Copyright © 2014, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *SAS® Macro Programming Made Easy, Third Edition.*
Full book available for purchase here.

Index

A

%ABORT statement 153*t*
ALL option 43*t*
ALL value 75–76
ampersand (&) 7, 24, 60–61, 64–65, 136*t*, 137*t*, 180, 232
AND (&) operator 155*t*
arithmetic expressions 155–156
ATTRC function 280
ATTRIB statement 203, 228
ATTRIBUTE variable 270–272
ATTRN function 18, 145–147, 262–264, 280, 346*t*
AUTHORREPORT macro program 303–306, 352*t*
autocall facility
  about 248
  resolving macro program references when using 258–259
  saving macro programs with 248–254
  using under UNIX and z/OS systems 252–254
autocall library
  about 19
  creating 248–250
  identifying 251–252
  making available to your programs 250–251
autocall macro programs 147–150, 198–200, 251–252
AUTOMATIC option 43*t*
AVERAGE macro variable 214–216

B

Base SAS application 12*t*
BOOLEAN conversion type 133–134*t*, 156
BOX= option 106–108
%BQUOTE function 136*t*, 178, 180–181, 187–188, 348*t*
branching, in macro processing 174–176

C

calculations, category of errors 277*t*
CALL EXECUTE
  about 202*t*, 216–226
  calling macro programs with 220–223
  calling specific macro programs with 223–226
  timing of 217–218, 219–221
CALL SYMDEL 202*t*
CALL SYMPUT 202*t*
CALL SYMPUTX 98, 202*t*, 213–214, 214–216
catalog names, concatenating with macro variables 58–59
CATALOG procedure 73, 255, 345*t*
CATS function 223–226
CEIL conversion type 134*t*
character data, editing for comparisons 264–266
character expressions, using resolution of as arguments 207–209
CHECKSURVEY macro program 148–150, 346*t*
CHECKVARNAME macro program 143–144, 346*t*
CLASS statement 352*t*
CLASS_STRING parameter 289–292
CLASSVAR macro variable 289–292
CLOSE language function 18
CMD option 69*t*
%CMPRES function 148*t*, 265–266
%CMPRES macro program 148
colon (:) 44, 174, 232
command-style macros 69*t*
commas, preventing interpretation of as argument delimiters 183–184
COMP2VARS macro program 158–160, 347*t*
compiler, defined 22
COMPRESS
  editing data set characters 265–266
  masking results from applying SAS language functions 199–200
concatenating
  catalog names with macro variables 58–59
  permanent SAS data set names with macro variables 58–59
concatenation (||) operators 154
conditional iteration 170–174
conditional processing 13–15, 158–167
CONTENTS procedure 341
%COPY statement 152*t*, 256, 257–258

D

DAILY macro program 343*t*
dash (-) 44

DATA= option, SGPLOT procedure 18
data set names
  concatenating with macro variables 58–59
  referencing 59
data sets
  characteristics of 261–264, 269–272
  sample 341
  using macro programs to execute PROC steps on multiple 5
  using variables as arguments to SYMGET function 204–206
DATA statement
  iterative processing of SAS steps 15
  in MAKESETS macro program 343*t*
DATASETS procedure
  CALL EXECUTE routine 219–221
  in DETAIL macro program 347*t*
  in LISTLIBRARY macro program 349*t*
  using %GOTO with 174–176
%DATATYP function 148*t*
date, formatting in TITLE statement with %SYSFUNC function 142
DATE() function 346*t*
debugging SAS programs 280–281, 309–316
DES= option 69*t*, 256
DETAIL macro program 347*t*
dictionary tables, displaying macro option settings with 241–243
DIFFRATE macro variable 286–289
%DISPLAY statement 154*t*
%DO loops
  about 15
  building PROC steps with iterative 167–169
  building SAS statements within steps with iterative 169–170
  creating macro programs 325–328
  iterative 153*t*
  in LISTSQLPUB macro program 350*t*
  in MULTREP macro program 347*t*
  in SALES macro program 343*t*
  SQL procedure 239–241
  writing in macro language 167–170
%DO statement 153*t*, 156, 277*t*
%DO %UNTIL loop
  errors and 289–292
  in MOSALES macro program 347*t*
  in TABLES macro program 352*t*
%DO %UNTIL statement 90, 156, 170–172
%DO %WHILE statement 153*t*, 156, 172–174, 347*t*
domains 110–114, 138–140
DSNAME parameter 145–147, 174–176, 262–264, 269–272
DSREPORT macro program 343*t*

E

%ELSE statement 15, 158, 292–295, 294–295, 343*t*
%ELSE-%IF statement 287–289
encrypting stored compiled macro programs 258
%END statement 76, 153*t*, 277*t*, 284–298, 290–292
EQ (=) operator 155*t*
equal sign (=) 82
ERROR: option 43*t*, 44, 76–77
errors
  about 273
  categorizing 275–278
  error checking 275–278, 298–306
  examples in macro programming 281–298
  minimizing in developing SAS programs that contain macro language 274–275
  tools for debugging macro programming 278–281
  types of 273–274
%EVAL function 133–135, 144–145, 154, 155–156, 277*t*, 285–289
execution error 274
existence, determining of 138–140
EXPORT procedure 352*t*
EXTFILES macro program 292–295, 352*t*

F

FACTS macro program 269–272, 351*t*
FCMP procedure 130
filerefs, defining under Windows 251–252
FINDPUBLISHER macro variable 232–234
FIRSTDATE macro variable 234–236
FIRSTPRICE macro variable 234–236
FIRSTTITLE macro variable 234–236
FLOOR conversion type 134*t*
FONT= attribute 191–193
FOOTNOTE statement
  about 20, 344*t*

masking special characters and mnemonic operators 193–198
modifying and selecting statements 163–165
in MYPAGES macro program 348*t*
preventing resolution of special characters in macro variable values 189
in RTF_END macro program 351*t*
in STANDARDOPTS macro program 343*t*

FREQ procedure
about 344*t*, 349*t*
creating macro variables with CALL SYMPUTX 213–214
displaying macro variables 46
placing text after macro variables 58
referencing permanent SAS data set names and macro variables 59
RESOLVE function 229
reviewing processing messages 301–303
TABLES statement 39–40
TITLE statement 39–40

functions 137–140, 137t, 261–266
See also specific functions

G

GE (>) operator 155*t*
GETOPT macro program 346*t*
GETOPTION function 142–143, 346*t*
GETPUB parameter 283–298
GETSALENAME macro program 350*t*
GETSECTION parameter 283–298
GLBSUBSET macro variable 106–108
global macro symbol tables 95, 97–101, 97*f*
global macro variables 98–101, 108–110
GLOBAL option 43*t*
%GLOBAL statement 97, 106–108, 110–114, 152*t*, 345*t*
%GOTO statement 153*t*, 174, 275*t*, 280, 347*t*
GT (~=) operator 155*t*

H

hard-coded programming constants, removing 317–320
HIGHREPORT macro program 223–226, 349*t*

I

%IF statement 76, 118, 123, 156, 157, 158, 211–212, 277*t*, 280–281, 294–295
%IF-%THEN statement 303–306, 325–328
%IF-%THEN-%DO statement 76
%IF-%THEN-%ELSE statement 90, 153*t*, 162–165, 166–167, 216–226, 347*t*
IN operator 154, 157, 166–167
IN (#) operator 155*t*
%INDEX function 130*t*
input stack 22, 116
%INPUT statement 154*t*
integer arithmetic 134
INTEGER conversion type 134*t*

interfaces
about 201–202
CALL EXECUTE routine 216–226
displaying macro option settings with SQL procedure and dictionary tables 241–243
macro variables created by SQL procedure 238–241
RESOLVE function 227–231
SYMGET function 202–209
SYMPUT call routine 209–216
SYMPUTX call routine 209–216
using macro facility features in SQL procedure 231–243

INTO clause 18, 98, 232–238, 239–241, 350*t*
iterations, conditional 170–174
iterative processing 15–16, 167–174

J

JCLDD statement 253–254

K

KEYPARM macro program 345*t*
KEYTEXT parameter 132–133
keyword parameters 80, 82–89

L

%label: statement 153*t*
LASTMSG macro program 301–303, 352*t*
LASTUPDATE macro variable 145–147

LE (<=) operator 155*t*
leading blanks 184–185
LEE macro variable 189
%LEFT function 148*t*
%LEFT macro program 148
%LENGTH function 130*t*
LENGTH statement 203, 228
%LET statement
    about 136, 152*t*, 344*t*
    creating macro programs 317–320
    creating macro variables 52–55
    data set characteristics 262–264
    in error checking 277*t*
    error tracing 287–289
    masking results from applying SAS language functions 199–200
    naming macro variables 111–114
    placing text before macro variable references 56
    program processing 28, 31
    quoting functions 179–180, 183–184, 184–185, 185–186, 186–187
    referencing macro variables 39–40, 106–108
    SQL procedure 242–243
    SYMGET function 204–206, 207–209
    using 149–150
    using %IF-%THEN-%ELSE statement to modify and select statements 163–165
library of utilities
    about 261
    building and saving routines 19
    programming routine tasks 266–272
    writing macro programs to behave like functions 261–266
LIBRARY= option 257–258
LISTAUTOMATIC macro program 349*t*
LISTINDEX macro variable 170–172
LISTLIBRARY macro program 219–221, 349*t*
LISTPARM macro program 345*t*
LISTSAMPLE macro program 262–264, 351*t*
LISTSQLPUB macro program 350*t*
LISTTEXT macro program 132–133, 346*t*
literal token 24
local macro symbol tables 96, 101–108
local macro variables 108–110
LOCAL option 43*t*
%LOCAL statement 110–114, 153*t*, 270–272
LOCLMVAR macro program 345*t*
LOCLMVAT macro program 111–114
logical expressions 156, 158–160, 278*t*
%LOWCASE function 148*t*
%LOWCASE macro program 148
LOWREPORT macro program 223–226, 349*t*
LT (<) operator 155*t*

M

MAC macro variable 214–216
macro activity, SAS processing without 22–23
macro character functions 130–133, 198–200
macro evaluation functions 133–135
macro expressions
    about 154–155
    arithmetic 155–156
    constructing 154–157
    logical 156
    IN operator 157
%MACRO function 254–258, 277*t*
macro functions
    category of errors 277*t*
    debugging macro programming with 280
%MACRO KEYPARM statement 83–86
macro language
    conditional processing with 158–167
    interfacing with SAS language functions 18–19
    iterative processing with 167–174
    processing SAS programs that contain 27–36
    SAS language *vs.* 277*t*
    selecting SAS steps for processing with 160–162
    writing iterative %DO loops in 167–170
macro language functions
    about 129–130
    autocall macro programs 147–150
    macro character functions 130–133
    macro evaluation functions 133–135
    macro quoting functions 135–137
    macro variable attribute functions 137–140
    %QSYSFUNC 141–147
    %SYSFUNC 141–147
macro language processing, compared with SAS language processing 26–27
macro language statements
    See also statements
    about 151–154
    compared with language statements 52

debugging macro programming with 279–280
types of 152
used in open code or inside macro programs 152–153*t*
used only inside macro programs 153–154*t*
used to display windows and prompt users for input 154*t*
using IN operator in 157
MACRO option 279*t*, 281–298
macro parameters 79–93, 277*t*
macro processing 21, 174–176
See also processing
macro processor 22, 109*f*, 110*f*, 117*f*, 118
macro program resolution category of errors 276*t*
macro programming
error examples in 281–298
tools for debugging 278–281
macro programs
See also specific macro programs
about 67–68, 247–248
autocall 147–150, 198–200, 251–252
calling with CALL EXECUTE routine 220–223, 223–226
calling with RESOLVE function 230–231
compiling 114–119
creating 68–73, 255–256
defining that accept varying number of parameter values 89–93
defining with keyword parameters 83–86, 86–89
defining with PARMBUFF option 90–93
defining with positional parameters 80–82, 86–89
demonstrating 71–73
displaying messages about processing of in SAS Log 77–79
displaying notes about compilation of in SAS Log 75–77
executing 74–75, 120–128
generalizing 325–328
passing values to through macro parameters 79–93
processing 114–128
referencing global macro variables in open code and from within 98–101
referencing macro variables defined as parameters to 102–108
refining 325–328
resolving references when using autocall facility 258–259
saving with autocall facility 248–254
saving with stored compiled macro facility 254–258
specifying parameters in 80–89, 189–198
in this book 343–353
tracing execution of with MLOGIC option 78–79
using to execute PROC steps on multiple data sets 5
writing 261–266, 307–338
macro quoting functions 135–137, 136–137*t*
%MACRO statement
about 68, 68*t*, 152*t*
building SAS steps with %DO %UNTIL loops 170–172
category of errors 277*t*
creating stored compiled macro programs 255–256
encrypting stored compiled macro programs 258
macro parameter names specified on 79
IN operator 157
in REPTITLE macro program 350*t*
saving and retrieving source code of stored compiled macro programs 257–258
specifying keyword parameters in macro programs 82
specifying positional parameters 80–82
macro symbol tables 22, 95–97
macro triggers 22, 24, 185–186, 187
macro variable attribute functions 137–140, 137*t*
macro variable resolution category of errors 276*t*
macro variable values
applying SAS statistical functions to with %SYSFUNC function 144–145
converting to uppercase with %UPCASE 132–133
extracting Nth item from with %SCAN 131–132
extracting text from with %SUBSTR 131
obtaining with RESOLVE function 228–229
preventing resolution of special characters in 188–189
retrieving with SYMGET function 206–207
submitting %PUT statement to display 45–46
macro variables
See also specific variables

about 37, 38
assigning results to 142–143
automatic 48–51
basic concepts of 38–39
combining with text 55–56
concatenating catalog names with 58–59
concatenating permanent data set names with 58–59
created by SQL procedure 238–241
creating with CALL SYMPUTX 213–214, 214–216
creating with INTO clause 236–237
creating with %LET statement 52–55
creating with SQL procedure 232–238
debugging macro programming with 280–281
defining 6–11
defining domains of 110–114
determining domain and existence of with macro variable attribute functions 138–140
displaying values 42–48
global 98–101, 108–110
illustrating resolution of references when combining them 61–63
naming 111–114
referencing 39–41, 59, 60–65, 102–108
replacing hard-coded programming constants with 317–320
resolution and use of quotation marks 41–42
resolving multiple ampersands preceding 64–65
saving summarizations in 232–234
selecting observations to process with 5
storing all values in 237–238
storing unique values in 237
updating with SQL procedure 232–238
user-defined 51–55
using 6–11
MAKEDS macro program 98–101, 102–108, 345*t*, 346*t*
MAKERTF macro program 292–295, 352*t*
MAKESETS macro program 343*t*
MAKESPREADHSEET macro program 292–295, 352*t*
MAR macro program 198, 348*t*
MARKUP macro program 285–289
masking
macro triggers 187
mnemonic operators 193–198
special characters 190–191, 193–198
MAUTOSOURCE option 250–251, 253–254
MAX operator 154
MCOMPILENOTE option 75–77
MEAN function 144–145
MEANS procedure
about 344*t*
building SAS steps with %DO %UNTIL loops 170–172
building SAS steps with %DO %WHILE loops 173–174
CALL EXECUTE routine 220–223, 223–226
CALL SYMPUTX 211–212, 214–216
conditional processing of SAS steps 13–15
defining macro programs with keyword parameters 85
defining macro programs with positional parameters 80–82
defining macro variables 40
extracting text from macro variable values with %SUBSTR 131
interfacing macro language and SAS language functions 18–19
in LISTPARM macro program 345*t*
in MOSALES macro program 347*t*
OTHROPTS parameter 86–89
preventing resolution of special characters in macro variable values 188–189
referencing macro variables 40
RESOLVE function 230–231
in SALES macro program 343*t*
STATS parameter 86–89
in STATSECTION macro program 349*t*
TITLE statement 41–42
WHERE statement 64, 131
%MEND statement 68, 70*t*, 127, 152*t*, 277*t*, 283–298
MERROR option 276*t*, 279*t*, 281–298
messages, displaying about macro program processing in SAS Log 77–79
MIN operator 154
MIN variable 214–216
MINDELIMITER option 69*t*, 157, 166–167
MINOPERATOR option 69*t*, 154
MINSALEPRICE parameter 299–301
MISSING option 87
MIXDPARM macro program 345*t*
MLOGIC option

creating macro programs 328
in debugging macro programming 278, 278*t*
displaying messages 77
errors and 292–295
executing REPORT macro program 329–331, 331–335, 335–338
tracing errors in expression evaluation with 285–289
tracing execution of macro programs with 78–79
mnemonic operators
about 177–178
category of errors 277*t*
masking 193–198
preventing interpretation of 187–188
specifying parameters that contain 189–198
MONTHS macro variable 131–132
MONTHSOLD parameter 299–301
MOSALES macro program 170–172, 347*t*
MOSECTDETAIL macro program 190–191, 348*t*
MOYRSALES macro program 172–174, 347*t*
MPRINT option
CALL EXECUTE routine 221–223
calling macro programs with CALL EXECUTE 224–226
creating macro programs 321–325, 328
in debugging macro programming 278, 279*t*
displaying messages 77
displaying SAS statements with 77–78
errors and 290–292, 296–298
executing REPORT macro program 329–338
MSTORED option 254–255
MULTCOND macro program 262–264, 351*t*
MULTREP macro program 167–169, 347*t*
MVS Batch, using autocall facilities under 253–254
MYPAGES macro program 193–198, 348*t*
MYREPORT macro program 118, 119*f*, 120–128, 122*f*, 127
MYSETTING macro variable 242–243
MYSQLSTEP macro variable 182–183

N

NAMES macro variable 186–187
names token 24
naming macro variables 39, 111–114
NE (^=) operator 154, 155*t*
NOAUTOCALL value 75–76
NOMINOPERATOR option 69*t*
NOMPRINT option 296–298
NONE value 75–76
NOSECURE option 70*t*
NOT (¬) operator 154, 155*t*
NOTE: option 43*t*, 44
notes, displaying about macro program compilation in SAS Log 75–77
NOTNAME function 143–144
%NRBQUOTE function 136*t*, 181
%NRQUOTE function 136*t*, 181
%NRSTR function
about 135–136, 136*t*, 178, 180–181, 348*t*
masking macro triggers with 185–186
masking results from applying SAS language functions 199–200
masking special characters 190–191
NSOLD macro variable 232–234
numbers token 24
numeric data set variables, creating with SYMGET function 206–207
NUMOBS= option 174–176
NVALID function 143–144

O

open code
macro variables in 38
referencing global macro variables in 98–101
OPEN language function 18
options 278–279
See also specific options
OPTIONS statement 20, 157, 166–167, 250–252, 254, 343*t*
</option(s)> element 69*t*, 70*t*
OPTS parameter 80–82, 83–86
OPTVALUE macro variable 142–143
OR (|) operator 155*t*
OTHROPTS parameter 86–89

P

parameter values
evaluating 299–301
masking special characters and mnemonic operators in 193–198

preventing misinterpretation of special characters in 191–193
<parameter-list> option, %MACRO element 68*t*
parameters
See also specific parameters
keyword 80, 82–89
referencing macro variables defined as parameters to macro programs 102–108
specifying 189–198
PARMBUFF option 69*t*, 80, 89–93, 170–172, 345*t*, 347*t*
PBUFF option 69*t*
PBUFFARMS macro program 345*t*
percent sign (%) 7, 24, 136*t*, 137*t*, 174, 180, 186–187
periods (.) 55
positional parameters 80–82, 86–89
PREFIX macro variable 57
PRINT procedure
about 5
in AUTHORREPORT macro program 352*t*
INTO clause 234–236
compiling macro programs 118
data set characteristics 262, 264
in DETAIL macro program 347*t*
documenting data set characteristics 269–272
evaluating parameter values 299–301
executing macro programs 125, 126*f*, 127
in LISTSAMPLE macro program 351*t*
in LISTTEXT macro program 346*t*
in MAKERTF macro program 352*t*
masking special characters 190–191
in MOSECTDETAIL macro program 348*t*
quoting functions 179–180
SYMGET function 205–206
using automatic variables 50–51
using %GOTO with 174–176
using macro language to select SAS steps for processing 160–162
PRINT10 macro program 281–298, 352*t*
processing
conditional 13–15, 158–167
iterative 15–16, 167–174
macro programs 114–128
reviewing messages 301–303
SAS programs that contain macro language 27–36
vocabulary of 21–22
without macro activity 22–23
PROFITCHART macro program 345*t*
program option, %MACRO element 68*t*
program steps, passing information between 16–18
programming routine tasks 266–272
PROJCOST macro program 352*t*
PUBLISHER macro variable 136
PUBLISHER parameter 166–167, 299–301
PUBLISHERREPORT macro program 162–165, 347*t*
PUBLISHERSALES macro program 191–193, 348*t*
punctuation, category of errors 275*t*
purpose, of macro variables 39
PUT function 211–212
%PUT LOCAL statement 214–216
%PUT statement
about 152*t*, 346*t*
applying quoting functions 182–189
CALL EXECUTE routine 219–221
in CHECKVARNAME macro program 346*t*
in COMP2VAR macro program 347*t*
debugging macro programming 279–280
displaying macro variable values with 43–46
displaying macro variables 42
%EVAL evaluation function examples 134–135*t*
in LISTAUTOMATIC macro program 349*t*
masking results from applying SAS language functions 199–200
quoting functions 183–186
reviewing processing messages 302–303
SQL procedure 232–234, 237
submitting to display text and macro variable values 45–46
%SYSEVALF evaluation function examples 134–135*t*
tracing errors in expression evaluation with 285–289
tracing problems at execution 287–289
using 142–143, 144–145, 148–150
using with %LET statement 52–54
%PUT_AUTOMATIC_ statement 44, 217–218
%PUT_GLOBAL_ statement 45
PUTN function 18, 145–147

Q

%QCMPRES function 178
%QLEFT function 178

%QLOWCASE function 178
%QSCAN function 130, 156, 178, 186–187, 348*t*
%QSUBSTR function 130, 156, 178, 200, 348*t*
%QSYSFUNC function 141–147, 141*f*, 178, 199–200, 280, 348*t*
%QTRIM function 178
quotation marks 24, 41–42, 186–187
%QUOTE function 136*t*, 181
quoting functions
- about 178
- applying 182–189
- commonly used 180–181
- how they work 181–182
- macro 135–137, 136–137*t*
- need for 179–180

quoting versions, using of macro character functions and autocall macro programs 198–200
%QUPCASE function 130, 178, 265–266

R

REP16K macro program 220–223, 349*t*
REPGRP macro variable 28, 31, 33, 34
REPMONTH parameter 172–174
REPORT macro program
- about 353*t*
- enhancing 338
- executing 328–338
- in PUBLISHERSALES macro program 348*t*
- in PUBLISHREPORT macro program 347*t*
- refining and generalizing macro programs 326–328
- in REP16K macro program 349*t*

REPORT procedure
- CALL EXECUTE routine 220–223, 223–226
- preventing misinterpretation of special characters 191–193
- in RTF_START macro program 351*t*
- standardizing RTF output 266–269, 267–269
- using %IF-%THEN-%ELSE statement to modify and select statements 162–165

REPORTA macro program 324–325, 353*t*
REPORTB macro program 324–325, 353*t*
REPORTC macro program 353*t*
REPORTS macro program 160–162, 347*t*
REPORTTITLE macro variable 185–186
REPTITLE macro program 39–40, 350*t*
REPTYPE parameter 160–162, 162–165
REPVAR macro variable 39–40
RESOLVE function
- about 202*t*, 227–231
- ATTRIB statement 228
- in error checking 276*t*
- FREQ procedure 229
- in GETSALENAME macro program 350*t*
- LENGTH statement 228
- obtaining macro variable values with 228–229
- using to call macro programs within DATA steps 230–231

RESPONSE parameter 148–150, 346*t*
RETAIN statement 352*t*
%RETURN statement 154*t*
RTF output, standardizing 266–269
RTF_END macro program 267–269, 351*t*
RTF_START macro program 267–269, 351*t*
RUN statement 26, 35

S

SALES macro program 343*t*
SAS catalogs, identifying autocall libraries stored in 252
SAS Component Language application 12*t*
SAS language
- functions 18–19, 142–143
- macro language *vs.* 277*t*
- processing, compared with macro language processing 26–27

SAS Log
- displaying messages about macro program processing in 77–79
- displaying notes about macro program compilation in 75–77

SAS macro facility
- about 4–5
- advantages of 5–11
- examples of 12–20
- where it can be used 11–12

SAS Macro Language: Reference 39, 48, 68, 68*t*, 69*t*, 70*t*, 152
SAS programs
- debugging 309–316
- processing 27–36
- testing 309–316

tokenizing 24–26
writing 309–316
SAS steps
conditional processing of 13–15
iterative processing of 15–16
SASAUTO= option
about 20
autocall libraries 251
maintaining access to autocall macro programs 251–252
SAS/CONNECT application 12*t*
SAS/GRAPH application 12*t*
SASMSTORE option 254–255
%SCAN function
about 130, 130–131*t*, 156, 346*t*
building SAS steps with %DO %UNTIL loops 170–172
creating macro programs 318–320
defining macro programs with PARMBUFF option 90
errors and 289–292
extracting Nth item from macro variable values 131–132
quoting functions 186–187
SECTION parameter 120–128, 214–216, 220–223
SECURE option 70*t*, 256, 258
SELECT statement 98, 232–241, 310–316
SELECTTITLES macro program 299–301, 352*t*
semicolon, preventing interpretation of as a SAS statement terminator 182–183
SENTENCE macro variable 54
SERROR option 276*t*, 279*t*, 281–298
SET statement 284–298
SGPANEL procedure 343*t*
SGPLOT procedure
building PROC steps with iterative %DO loops 167–169
creating macro programs 310–316, 318–320, 325–328
DATA= option 18
DATA step 74–75
defining macro programs with PARMBUFF option 90–93
enhancing REPORT macro program 338
executing REPORT macro program 331–335
in MULTREP macro program 347*t*
in PROFITCHART macro program 345*t*
in REPORTB macro program 353*t*
TITLE statement 16–18
source code 257–258
SOURCE option
creating stored compiled macro programs 255–256
</option(s)> element 70*t*
saving and retrieving source code of stored compiled macro programs 257–258
special characters
about 177–178
category of errors 277*t*
masking 190–191, 193–198
preventing misinterpretation of 191–193
preventing resolution of in macro variable values 188–189
specifying parameters that contain 189–198
special token 24
SQL procedure
about 50, 348*t*, 350*t*
in AUTHORREPORT macro program 352*t*
computing sum of scales with 18
creating macro programs 310–316, 318–320, 326–328
creating macro variables with 232–238
default action of INTO clause 234–236
displaying macro option settings with 241–243
documenting data set characteristics 270–272
executing REPORT macro program 331–335
%LET statement 242–243
macro variables created by 238–241
%PUT statement 232–234, 237
quoting functions 182–183
in REPORT macro program 353*t*
saving summarizations in macro variables 232–234
SELECT statement 98, 232–241
SQLOBS automatic macro variable 239–241
storing all values in macro variables 237–238
storing unique values in macro variables with 237
updating macro variables with 232–238
using INTO clause to create macro variables 236–237
using macro facility features in 231–243
SQLEXITCODE macro variable 238*t*
SQLOBS macro variable 238*t*, 239–241

SQLOOPS macro variable 238*t*
SQLRC macro variable 238*t*
SQLXMSG macro variable 238, 239*t*
SQLXRC macro variable 238, 239*t*
SRC option 70*t*
standardizing RTF output 266–269
STANDARDOPTS macro program 343*t*
START parameter 80–82, 83–86, 86–89, 326–328
STATE macro variable 187–188
statements
  See also specific statements
  compared with SAS macro language statement 52
  displaying with MPRINT option 77–78
  modifying 162–165
  selecting 162–165
  transferring from input stack to word scanner 116
STATS parameter 86–89
STATSECTION macro program 214–216, 349*t*
STMT option 69*t*
STOP parameter 80–82, 83–86, 86–89, 326–328
STORE option 70*t*, 257–258
stored compiled facility 248
stored compiled macro programs 257–259
%STR function
  about 136*t*, 178, 180–181, 348*t*
  masking special characters 190–191
  masking unbalanced quotation marks and preceding percent signs 186–187
  preserving leading and trailing blanks 184–185
  preventing interpretation of commas as argument delimiters 183–184
  preventing interpretation of semicolon as SAS statement terminator with 182–183
  quoting functions 185–186
STYLE parameter 191–193
STYLEHEADER parameter 191–193
STYLEREPORT parameter 191–193
SUBSET macro variable 98–101, 102–108, 111–114
%SUBSTR function 130, 131, 131*t*, 156, 183–184, 200, 346*t*, 348*t*
SUMSOFTPROG macro variable 18
SUMYEARS macro program 169–170, 347*t*
%SUPERQ function 136*t*, 137*t*, 178, 180–181, 188–189, 265–266, 348*t*
SURVLIB macro variable 59
%SYMBLOBL macro variable 214–216
symbol tables, specifying 214–216
SYMBOLGEN option
  about 344*t*
  creating macro programs 321–325, 328
  debugging macro programming 278, 279*t*, 280
  displaying macro variables 42
  displaying messages 77
  enabling to display macro variable values 46–48
  quoting functions 182
  resolving multiple ampersands that precede macro variables 64
  tracing resolution of indirect macro variable references 63
%SYMDEL statement 102–108, 138–140, 152*t*, 346*t*
SYMEXIST tool 202*t*
%SYMEXLIST function 137*t*, 280
SYMGET function
  about 202–209, 202*t*, 349*t*
  creating numeric data set variables with 206–207
  retrieving macro variables with 206–207
  similarity to RESOLVE function 227
  using data set variables as arguments to 204–206
  using resolution of character expressions as arguments to 207–209
%SYMGLOBL function 137*t*, 280, 346*t*
%SYMLOCAL function 137*t*, 202*t*, 280, 346*t*
%SYMLOCAL macro variable 214–216
SYMPUT call routine 209–216
SYMPUTX call routine 16–18, 209–216
syntax error 274
%SYSCALL statement 152*t*
SYSDATE macro variable 49*t*, 131
SYSDATE9 macro variable 49*t*, 97–101
SYSDAY macro variable 39–40, 49*t*, 97–101, 125, 343*t*
SYSDEN macro variable 49*t*
SYSERR macro variable 49*t*
SYSERRORTEXT macro variable 49*t*, 301–303, 352*t*
%SYSEVALF function
  about 133–135, 154, 155–156, 346*t*
  in error checking 277*t*
  error tracing 285–289
  preventing interpretation of mnemonic operators 187–188
  using 144–145
%SYSEXEC statement 153*t*

SYSFILRC macro variable 49*t*
%SYSFUNC function
  about 141–142, 346*t*
  applying SAS language functions 145–147
  applying SAS statistical functions 144–145
  assigning results to macro variables 142–143
  data set characteristic 262–264
  debugging macro programming 280
  determining if values are valid SAS variable names with 143–144
  in DSREPORT macro program 343*t*
  executing SAS language functions with 142–143
  formatting date in TITLE statement with 142
  masking results from applying SAS language functions 199–200
  using 141–142, 141–147, 141*f*, 149–150
%SYSGET function 141*f*
SYSLAST macro variable 49*t*
SYSLIBRC macro variable 49*t*
%SYSLPUT statement 153*t*
%SYSMACDELETE statement 73
SYSMACRONAME macro variable 49*t*
SYSPBUFF macro variable 89–93
%SYSPROD function 141*f*
%SYSRPUT statement 153*t*
system information, displaying 12–13
system option settings category of errors 276*t*
SYSTIME macro variable 49*t*
SYSVER macro variable 49*t*, 97–101
SYSWARNINGTEXT macro variable 49*t*, 301–303, 352*t*

T

TABLES macro program 289–292, 352*t*
TABLES statement
  errors and 296–298
  FREQ procedure 39–40
  placing text after macro variable references 57–58
  in TABLES macro program 352*t*
TABULATE procedure
  about 3, 343*t*
  in AUTHORREPORT macro program 352*t*
  building PROC steps with iterative %DO loops 167–169
  editing data set characters 264–266
  enhancing REPORT macro program 338
  errors and 289–292, 296–298
  in MAKEDS macro program 345*t*
  in PROJCOST macro program 352*t*
  referencing macro variables defined as parameters to macro programs 103–108
  in REPORTA macro program 353*t*
  in REPORTS macro program 347*t*
  in TABLES macro program 352*t*
  in TRIMNAME macro program 351*t*
  using macro language to select SAS steps for processing 160–162
testing SAS programs 309–316
text
  combining macro variables with 55–56
  extracting from macro variable values with %SUBSTR 131
  placing after macro variable references 57–58
  placing before macro variable references 56
  unmasking 198
text values
  macro variables as 38–39
  submitting %PUT statement to display 45–46
<text> element 70*t*
timing, of CALL EXECUTE routine 217–218, 219–221
TITLE statement
  about 3, 133, 343*t*, 344*t*, 346*t*
  building SAS steps with %DO %UNTIL loops 171–172
  CALL SYMPUTX 214–216
  formatting date with %SYSFUNC function in 142
  FREQ procedure 39–40
  masking special characters and mnemonic operators 193–198
  MEANS procedure 41–42
  in MYPAGES macro program 348*t*
  passing information between program steps 16–18
  in REPTITLE macro program 350*t*
  resolving macro variables enclosed in quotation marks 41–42
  in RTF_END macro program 351*t*
  in STANDARDOPTS macro program 343*t*
  in STATSECTION macro program 349*t*
  using %UNQUOTE function 198

in VENDORTITLES macro program 347*t*
TITLE1 statement 20
TITLE2 statement 166–167
tokenizing SAS programs 24–26
tokens
about 23–24
defined 22
transferring to macro processor 117*f*, 118
tools, for debugging macro programming 278–281
TOTSALES macro variable 232–234
trailing blanks 184–185
%TRIM function 148*t*
TRIMNAME macro program 264–266, 351*t*

U

UNIX 20, 252–254
unmasking text 198
%UNQUOTE function 137*t*, 178, 198, 348*t*
%UPCASE function 130, 131*t*, 132–133, 148–150
%UPCASE macro program 346*t*
uppercase, converting macro variable values to with %UPCASE 132–133
USER option 43*t*
user-defined macro variables 38, 51

V

VALIDRESPONSES macro variable 149
VALUE variable 270–272
values
determining if they are valid SAS variable names 143–144
macro variable 42–48
passing to macro programs through macro parameters 79–93
VAR statement 179–180
variables
See specific variables
VARLIST parameter 174–176
VARNUM macro variable 290–292
VARS= parameter 326–328
VENDORTITLES macro program 166–167, 347*t*
%VERIFY function 148–150, 148*t*, 346*t*

W

WARNING: option 43*t*, 44
WHERE statement
about 133, 343*t*, 344*t*, 346*t*
building SAS steps with %DO %UNTIL loops 170–172
editing data set characters 264–266
errors and 283–298
evaluating parameter values 299–301
MEANS procedure 64, 131
in WHSTMT macro program 352*t*
WHEREVAR macro variable 64
WHSTMT macro program 283–298, 284–298, 352*t*
%WINDOW statement 154*t*
WONTWORK macro variable 179–180
word scanner 22, 26, 116

Y

YEARIDX macro variable 173–174
YEARLIST parameter 172–174
YEARSTRING macro variable 169–170

Z

z/OS systems, using autocall facilities under 252–254

Symbols

%* comment; statement 152*t*
- operator 155*t*
/ operator 155*t*
* operator 155*t*
** operator 155*t*
^ operator 155*t*
+ operator 155*t*
~ operator 155*t*
|| (concatenation) operators 154
>= operator 155*t*
& (ampersand) 7, 24, 60–61, 64–65, 136*t*, 137*t*, 180, 232
& (AND) operator 155*t*
: (colon) 44, 174, 232
- (dash) 44
= (EQ) operator 155*t*
= (equal sign) 82
> (GE) operator 155*t*

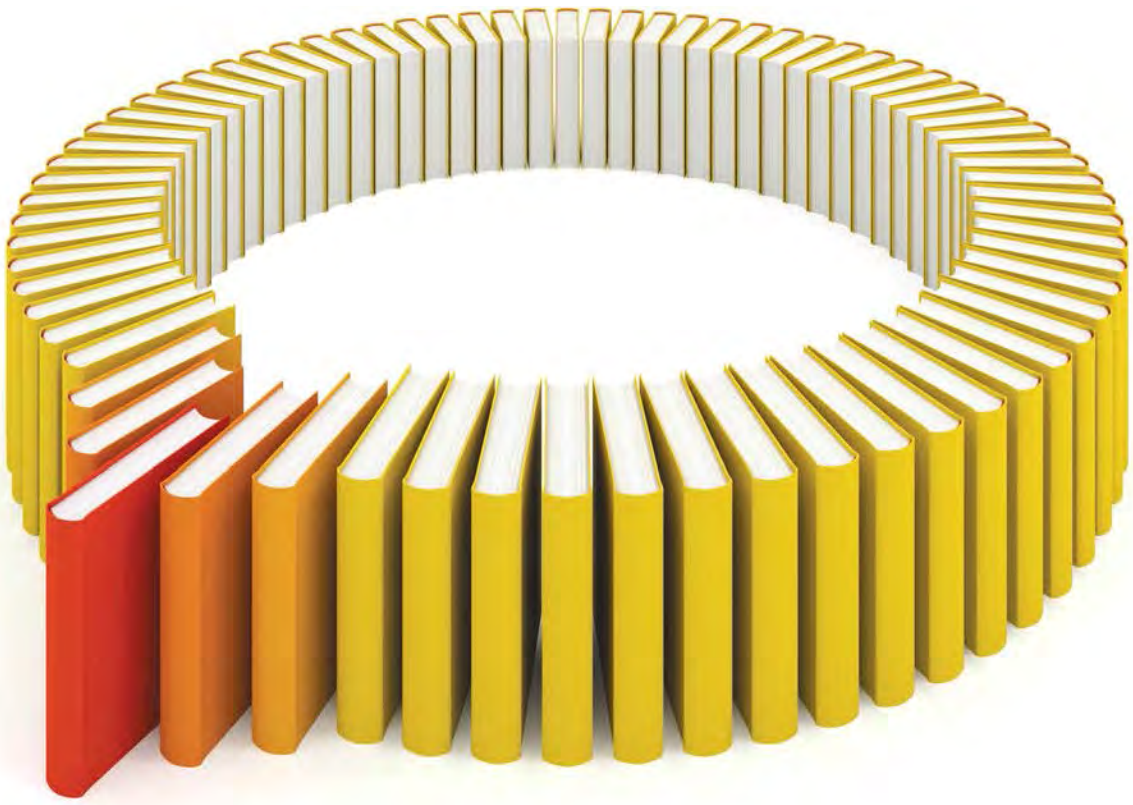
~= (GT) operator 155*t*
# (IN) operator 155*t*
<= (LE) operator 155*t*
< (LT) operator 155*t*
^= (NE) operator 155*t*
(¬) NOT operator 154, 155*t*
| (OR) operator 155*t*
% (percent sign) 7, 24, 136*t*, 137*t*, 174, 180, 186–187
. (periods) 55

From *SAS® Macro Programming Made Easy, Third Edition* by Michele M. Burlew. Copyright © 2014, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

About The Author

Michele M. Burlew designs and programs SAS applications for data management, data analysis, report writing, and graphics for academic and corporate clients. A SAS user since 1980, she has expertise in many SAS products and operating systems. Burlew is the author of seven SAS Press books: *SAS Hash Object Programming Made Easy; Combining and Modifying SAS Data Sets: Examples, Second Edition; Output Delivery System: The Basics and Beyond (coauthor)*; *SAS Guide to Report Writing: Examples, Second Edition; SAS Macro Programming Made Easy*, *Third Edition; Debugging SAS Programs: A Handbook of Tools and Techniques*; and *Reading External Data Files Using SAS: Examples Handbook*.

Learn more about this author by visiting her author page at: http://support.sas.com/burlew. There you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more.



Gain Greater Insight into Your SAS® Software with SAS Books.

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

support.sas.com/bookstore
for additional books and resources.

sas® THE POWER TO KNOW®

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613