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
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Displaying Macro Variable Values</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Using the %PUT Statement</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Displaying Macro Variable Values As They Resolve by Enabling the SYMBOLGEN System Option</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Understanding Automatic Macro Variables</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Understanding User-Defined Macro Variables</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Creating Macro Variables with the %LET Statement</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Combining Macro Variables with Text</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Placing Text before a Macro Variable Reference</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Placing Text after a Macro Variable Reference</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Concatenating Permanent SAS Data Set Names and Catalog Names with Macro Variables</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Referencing Macro Variables Indirectly</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>Chapter 4 Macro Programs</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Creating Macro Programs</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>Executing a Macro Program</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Displaying Notes about Macro Program Compilation in the SAS Log</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Displaying Messages about Macro Program Processing in the SAS Log</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Using MPRINT to Display the SAS Statements Submitted by a Macro Program</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Using the MLOGIC Option to Trace Execution of a Macro Program</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>Passing Values to a Macro Program through Macro Parameters</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Specifying Positional Parameters in Macro Programs</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Specifying Keyword Parameters in Macro Programs</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Specifying Mixed Parameter Lists in Macro Programs</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Defining a Macro Program That Can Accept a Varying Number of Parameter Values</td>
<td>89</td>
</tr>
<tr>
<td>5</td>
<td>Chapter 5 Understanding Macro Symbol Tables and the Processing of Macro Programs</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Understanding Macro Symbol Tables</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Understanding the Global Macro Symbol Table</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Understanding Local Macro Symbol Tables</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Working with Global Macro Variables and Local Macro Variables</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>Defining the Domain of a Macro Variable by Using the %GLOBAL or %LOCAL Macro Language Statements</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>Processing of Macro Programs</td>
<td>114</td>
</tr>
</tbody>
</table>
## How a Macro Program Is Compiled

How a Macro Program Executes

### Chapter 6 Macro Language Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>129</td>
</tr>
<tr>
<td>Macro Character Functions</td>
<td>130</td>
</tr>
<tr>
<td>Macro Evaluation Functions</td>
<td>133</td>
</tr>
<tr>
<td>Macro Quoting Functions</td>
<td>135</td>
</tr>
<tr>
<td>Macro Variable Attribute Functions</td>
<td>137</td>
</tr>
<tr>
<td>Other Macro Functions</td>
<td>141</td>
</tr>
<tr>
<td>SAS Supplied Autocall Macro Programs Used Like Functions</td>
<td>147</td>
</tr>
</tbody>
</table>

### Chapter 7 Macro Expressions and Macro Programming Statements

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>151</td>
</tr>
<tr>
<td>Constructing Macro Expressions</td>
<td>154</td>
</tr>
<tr>
<td>Understanding Arithmetic Expressions</td>
<td>155</td>
</tr>
<tr>
<td>Understanding Logical Expressions</td>
<td>156</td>
</tr>
<tr>
<td>Understanding the IN Operator As Used in Macro Language Statements</td>
<td>157</td>
</tr>
<tr>
<td>Conditional Processing with the Macro Language</td>
<td>158</td>
</tr>
<tr>
<td>Iterative Processing with the Macro Language</td>
<td>167</td>
</tr>
<tr>
<td>Writing Iterative %DO Loops in the Macro Language</td>
<td>167</td>
</tr>
<tr>
<td>Conditional Iteration with %DO %UNTIL</td>
<td>170</td>
</tr>
<tr>
<td>Conditional Iteration with %DO %WHILE</td>
<td>172</td>
</tr>
<tr>
<td>Branching in Macro Processing</td>
<td>174</td>
</tr>
</tbody>
</table>

### Chapter 8 Masking Special Characters and Mnemonic Operators

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>177</td>
</tr>
<tr>
<td>Why Are Quoting Functions Called Quoting Functions?</td>
<td>178</td>
</tr>
<tr>
<td>Illustrating the Need for Macro Quoting Functions</td>
<td>179</td>
</tr>
<tr>
<td>Describing the Commonly Used Macro Quoting Functions</td>
<td>180</td>
</tr>
<tr>
<td>Understanding How Macro Quoting Functions Work</td>
<td>181</td>
</tr>
<tr>
<td>Applying Macro Quoting Functions</td>
<td>182</td>
</tr>
<tr>
<td>Specifying Macro Program Parameters That Contain Special Characters or Mnemonic Operators</td>
<td>189</td>
</tr>
<tr>
<td>Unmasking Text and the %UNQUOTE Function</td>
<td>198</td>
</tr>
<tr>
<td>Using Quoting Versions of Macro Character Functions and Autocall Macro Programs</td>
<td>198</td>
</tr>
</tbody>
</table>
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

**Introduction**

Understanding the Types of Errors That Can Occur in Macro Programming

Minimizing Errors in Developing SAS Programs That Contain Macro Language

Categorizing and Checking for Common Problems in Macro Programming

Understanding the Tools That Can Debug Macro Programming

- Using SAS System Options to Debug Macro Programming
- Using Macro Language Statements to Debug Macro Programming
- Using Macro Functions to Debug Macro Programming
- Using Automatic Macro Variables to Debug Macro Programming

Examples of Solving Errors in Macro Programming

Improving Your Macro Programming by Including Error Checking

---

# Chapter 13 A Stepwise Method for Writing Macro Programs

**Introduction**

Building a Macro Program in Four Steps

Applying the Four Steps to an Example

- Step 1: Write, test, and debug the SAS program(s) that you want the macro program to build
- Step 2: Remove hard-coded programming constants from the program(s) in Step 1 and replace these constants with macro variables
- Step 3: Create macro program(s) from the program(s) in Step 2
- Step 4: Refine and generalize the macro program(s) in Step 3 by adding macro language statements like `%IF-%THEN` and `%DO` groups

Executing the REPORT Macro Program

Enhancing the Macro Program REPORT

---

# Part 3 Appendixes

**Appendix A Sample Data Set**

**Appendix B Reference to Examples in This Book**

**Index**
## 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

---

Full book available for purchase [here](#).
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%INDEX(source, string)</code></td>
<td>returns the position in <code>source</code> of the first character of <code>string</code>.</td>
</tr>
<tr>
<td>`%LENGTH(string</td>
<td>text expression)`</td>
</tr>
<tr>
<td><code>%SCAN(argument, n &lt;,charlist&lt;,modifiers&gt;&gt;)</code></td>
<td>searches for the <code>n</code>th word in <code>argument</code>, where <code>argument</code> can be text or an expression and the words in <code>argument</code> are separated by <code>charlist</code>. Specify <code>modifiers</code> to modify how <code>%SCAN</code> determines word boundaries in</td>
</tr>
</tbody>
</table>
Chapter 6 Macro Language Functions

**%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.

```plaintext
proc means data=books.ytdsales;
  title "Sales for 01%substr(&sysdate,3,3) through &sysdate9999;"
  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.

```plaintext
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.

```sas
%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:

```sas
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.

```sas
%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>
<thead>
<tr>
<th>Conversion Type</th>
<th>Result that is returned by %SYSEVALF</th>
</tr>
</thead>
</table>
| BOOLEAN         | 0 if the result of the expression is 0 or null  
                 | 1 if the result is any other value      |
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

<table>
<thead>
<tr>
<th>%PUT Statement</th>
<th>Results in SAS log</th>
</tr>
</thead>
<tbody>
<tr>
<td>%put %eval(33 + 44);</td>
<td>77</td>
</tr>
<tr>
<td>%put %eval(33.2 + 44.1);</td>
<td>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</td>
</tr>
<tr>
<td>%put %sysevalf(33.2 + 44.1);</td>
<td>77.3</td>
</tr>
<tr>
<td>%put %sysevalf(33.2 + 44.1,integer);</td>
<td>77</td>
</tr>
<tr>
<td>%let a=3; %let b=10; %put %eval(&amp;b/&amp;a);</td>
<td>3</td>
</tr>
<tr>
<td>%put %sysevalf(&amp;b/&amp;a);</td>
<td>3.3333333333333333</td>
</tr>
<tr>
<td>%put %sysevalf(&amp;b/&amp;a,ceil);</td>
<td>4</td>
</tr>
<tr>
<td>%put %sysevalf(&amp;b/&amp;a,floor);</td>
<td>3</td>
</tr>
<tr>
<td>%put %sysevalf(&amp;b/&amp;a,boolean);</td>
<td>1</td>
</tr>
</tbody>
</table>
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:

```sas
%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.

```sas
%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 \textit{Doe}\&\textit{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**

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BQUOTE(character-string</td>
<td>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 (%).</td>
</tr>
<tr>
<td>text expression)</td>
<td></td>
</tr>
<tr>
<td>%NRBQUOTE(character-string</td>
<td>Does the same as %BQUOTE and additionally masks ampersands (&amp;) and percent signs (%). However, SAS recommends that you use %SUPERQ instead of %NRBQUOTE.</td>
</tr>
<tr>
<td>text expression)</td>
<td></td>
</tr>
<tr>
<td>%QUOTE(character-string</td>
<td>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 (%).</td>
</tr>
<tr>
<td>text expression)</td>
<td></td>
</tr>
<tr>
<td>%NRQUOTE(character-string</td>
<td>Does the same as %QUOTE and additionally masks ampersands (&amp;) and percent signs (%). However, SAS recommends that you use %SUPERQ instead of %NRQUOTE.</td>
</tr>
<tr>
<td>text expression)</td>
<td></td>
</tr>
<tr>
<td>%STR(character-string)</td>
<td>Mask special characters and mnemonic operators in constant text at macro compilation.</td>
</tr>
<tr>
<td>%NRSTR(character-string)</td>
<td>Does the same as %STR and additionally masks ampersands (&amp;) and percent signs (%).</td>
</tr>
</tbody>
</table>
Macro Language Functions

Chapter 6 Macro Language Functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SYMEXIST(macro-variable-name)</td>
<td>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.</td>
</tr>
<tr>
<td>%SYMGLOBL(macro-variable-name)</td>
<td>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.</td>
</tr>
<tr>
<td>%SYMLOCAL(macro-variable-name)</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

```sas
%* 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=⊂
296     %* What is domain of SUBSET and GLBSUBSET inside MAKEDS?;
297     %put ******** Inside macro program;
298     %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;
%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

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

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;

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

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

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.
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.

```sas
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.

```sas
%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
```
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.

```sas
%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;
```
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.

```sas
%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.

```sas
%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.
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CMPRES(text</td>
<td>text expression)</td>
</tr>
<tr>
<td>%DATATYP(text</td>
<td>text expression)</td>
</tr>
<tr>
<td>%LEFT(text</td>
<td>text expression)</td>
</tr>
<tr>
<td>%LOWCASE(text</td>
<td>text expression)</td>
</tr>
<tr>
<td>%TRIM(text</td>
<td>text expression)</td>
</tr>
<tr>
<td>%VERIFY(source</td>
<td>excerpt)</td>
</tr>
</tbody>
</table>

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.

```sas
%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:

```sas
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
```

Index

A

%ABORT statement 153t
_ALL_ option 43t
ALL value 75–76
ampersand (&) 7, 24, 60–61, 64–65, 136t, 137t, 180, 232
AND (&) operator 155t
arithmetic expressions 155–156
ATTRC function 280
ATTRIB statement 203, 228
ATTRIBUTE variable 270–272
ATTRN function 18, 145–147, 262–264, 280, 346t
AUTHORREPORT macro program 303–306, 352t
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 43t
AVERAGE macro variable 214–216
timing of 217–218, 219–221
CALL SYMDEL 202t
CALL SYMPUT 202t
CALL SYMPUTX 98, 202t, 213–214, 214–216
catalog names, concatenating with macro variables 58–59
CATALOG procedure 73, 255, 345t
CATS function 223–226
CEIL conversion type 134t
character data, editing for comparisons 264–266
character expressions, using resolution of as arguments 207–209
CHECKSURVEY macro program 148–150, 346t
CHECKVARNAME macro program 143–144, 346t
CLASS statement 352t
CLASS_STRING parameter 289–292
CLASSVAR macro variable 289–292
CLOSE language function 18
CMD option 69t
%CMPRES function 148t, 265–266
%CMPRES macro program 148
colon (:) 44, 174, 232
command-style macros 69t
commas, preventing interpretation of as argument delimiters 183–184
COMP2VARS macro program 158–160, 347t
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 152t, 256, 257–258

dash (-) 44

B

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

calculations, category of errors 277t
CALL EXECUTE
  about 202t, 216–226
  calling macro programs with 220–223
  calling specific macro programs with 223–226

C

calculations, category of errors 277t
CALL EXECUTE
  about 202t, 216–226
  calling macro programs with 220–223
  calling specific macro programs with 223–226

D

DAILY macro program 343t
Index

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  343t

DATASETS procedure
  CALL EXECUTE routine  219–221
  in DETAIL macro program  347t
  in LISTLIBRARY macro program  349t
  using %GOTO with  174–176

%DATATYP function  148t

date, formatting in TITLE statement with %SYSFUNC function  142

DATE() function  346t

debugging SAS programs  280–281, 309–316

DES= option  69t, 256

DETAIL macro program  347t
dictionary tables, displaying macro option settings with  241–243
differrate macro variable  286–289

%DISPLAY statement  154t

%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  153t
  in LISTSQLPUB macro program  350t
  in MULTREP macro program  347t
  in SALES macro program  343t
  SQL procedure  239–241
  writing in macro language  167–170

%DO statement  153t, 156, 277t

%DO %UNTIL loop
  errors and  289–292
  in MOSALES macro program  347t
  in TABLES macro program  352t

%DO %UNTIL statement  90, 156, 170–172

%DO %WHILE statement  153t, 156, 172–174, 347t
domains  110–114, 138–140


DSREPORT macro program  343t

E

%ELSE statement  15, 158, 292–295, 294–295, 343t
%ELSE-%IF statement  287–289

encrypting stored compiled macro programs  258

%END statement  76, 153t, 277t, 284–298, 290–292

EQ (=) operator  155t
equal sign (=)  82

ERROR: option  43t, 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, 277t, 285–289

execution error  274

existence, determining of  138–140

EXPORT procedure  352t

EXTFILES macro program  292–295, 352t

F

FACTS macro program  269–272, 351t

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  134t

FONT= attribute  191–193

FOOTNOTE statement
  about  20, 344t
masking special characters and mnemonic operators 193–198
modifying and selecting statements 163–165
in MYPAGES macro program 348
preventing resolution of special characters in macro variable values 189
in RTF_END macro program 351
in STANDARDOPTS macro program 343
FREQ procedure
about 344, 349
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, 137, 261–266
See also specific functions

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

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

I
%IF statement 76, 118, 123, 156, 157, 158, 211–212, 277, 280–281, 294–295
%IF-%THEN statement 303–306, 325–328
%IF-%THEN-%DO statement 76
%IF-%THEN-%ELSE statement 90, 153, 162–165, 166–167, 216–226, 347
IN operator 154, 157, 166–167
IN (#) operator 155
%INDEX function 130
%INPUT statement 154
integer arithmetic 134
INTEGER conversion type 134
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
iterations, conditional 170–174
iterative processing 15–16, 167–174

J
JCLDD statement 253–254

K
KEYPARAM macro program 345
KEYTEXT parameter 132–133
keyword parameters 80, 82–89

L
%label: statement 153
LASTMSG macro program 301–303, 352
LASTUPDATE macro variable 145–147
LE (<=) operator 155
leading blanks 184–185
LEE macro variable 189
%LEFT function 148
%LEFT macro program 148
%LENGTH function 130
LENGTH statement 203, 228
%LET statement
about 136, 152, 344
creating macro programs 317–320
creating macro variables 52–55
data set characteristics 262–264
in error checking 277
error tracing 287–289
masking results from applying SAS language functions 199–200
naming macro variables 111–114
placer text before macro variable references 56
program processing 28, 31
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
LISTINDEX macro variable 170–172
LISTLIBRARY macro program 219–221, 349
LISTPARM macro program 345
LISTSAMPLE macro program 262–264, 351
LISTSQLPUB macro program 350
LISTTEXT macro program 132–133, 346
literal token 24
local macro symbol tables 96, 101–108
local macro variables 108–110
_LOCAL_ option 43
%LOCAL statement 110–114, 153, 270–272
LOCLMVAT macro program 111–114
logical expressions 156, 158–160, 278
%LOWCASE function 148
%LOWCASE macro program 148
LOWREPORT macro program 223–226, 349
LT (<) operator 155
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
macro functions
category of errors 277
debbuging 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
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
used only inside macro programs 153–154
used to display windows and prompt users for input 154
using IN operator in 157
MACRO option 279, 281–298
macro parameters 79–93, 277
macro processing 21, 174–176
See also processing
macro processor 22, 109f, 110f, 117f, 118
macro program resolution category of errors 276
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
%MACRO statement
about 68, 68t, 152
building SAS steps with %DO %UNTIL loops 170–172
category of errors 277
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
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
macro variable resolution category of errors 276
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 237–238
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
MAKERFT macro program 292–295, 352
MAKESETS macro program 343
MAKESPREADHSEET macro program 292–295, 352
MAR macro program 198, 348
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
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
in MOSALES macro program 347
OTHOPTS 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
STATS parameter 86–89
in STATSECTION macro program 349
TITLE statement 41–42
WHERE statement 64, 131
%MEND statement 68, 70, 127, 152, 277, 283–298
MERROR option 276
messages, displaying about macro program processing in SAS Log 77–79
MIN operator 154
MIN variable 214–216
MINDELIMITER option 69, 157, 166–167
MINOPERATOR option 69, 154
MINSALEPRICE parameter 299–301
MISSING option 87
MIXDPARM macro program 345
MLOGIC option
creating macro programs 328
in debugging macro programming 278, 279t
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 277t
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, 347t
MOSECTDETAIL macro program 190–191, 348t
MOYRSALES macro program 172–174, 347t

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, 279t
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, 351t
MULTREP macro program 167–169, 347t
MVS Batch, using autocall facilities under 253–254
MYPAGES macro program 193–198, 348t
MYREPORT macro program 118, 119f, 120–128, 122f, 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, 155t
NOAUTOCALL value 75–76
NOMINOPERATOR option 69t
NOMPRINT option 296–298
NONE value 75–76
NOSECURE option 70t
NOT (¬) operator 154, 155t
NOTE: option 43t, 44
notes, displaying about macro program compilation in SAS Log 75–77
NOTNAME function 143–144
%NRBQUOTE function 136t, 181
%NRQUOTE function 136t, 181
%NRSTR function
about 135–136, 136t, 178, 180–181, 348t
masking macro triggers with 185–186
masking special characters 190–191
masking results from applying SAS language functions 199–200
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
reversing global macro variables in 98–101

OPTIONS statement 20, 157, 166–167, 250–252, 254, 343t

OPTS parameter 80–82, 83–86
OPTVALUE macro variable 142–143
OR (|) operator 155t

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
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 69t, 80, 89–93, 170–172, 345t, 347t
PBUFF option 69t
PBUFFARMS macro program 345t
percent sign (%) 7, 24, 136t, 137t, 174, 180, 186–187
periods (.) 55
positional parameters 80–82, 86–89
PREFIX macro variable 57
PRINT procedure
about 5
in AUTHORREPORT macro program 352t
INTO clause 234–236
compiling macro programs 118
data set characteristics 262, 264
in DETAIL macro program 347t
documenting data set characteristics 269–272
evaluating parameter values 299–301
executing macro programs 125, 126f, 127
in LISTSAMPLE macro program 351t
in LISTTEXT macro program 346t
in MAKERTF macro program 352t
masking special characters 190–191
in MOSECTDETAIL macro program 348t
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, 352t
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 345t
program option, %MACRO element 68t
program steps, passing information between 16–18
programming routine tasks 266–272
PROJCOST macro program 352t
PUBLISHER macro variable 136
PUBLISHER parameter 166–167, 299–301
PUBLISHERREPORT macro program 162–165, 347t
PUBLISHERSALES macro program 191–193, 348t
punctuation, category of errors 275t
purpose, of macro variables 39
PUT function 211–212
%PUT LOCAL statement 214–216
%PUT statement
about 152t, 346t
applying quoting functions 182–189
CALL EXECUTE routine 219–221
in CHECKVARNAMe macro program 346t
in COMP2VAR macro program 347t
debugging macro programming 279–280
displaying macro variable values with 43–46
displaying macro variables 42
%EVAL evaluation function examples 134–135t
in LISTAUTOMATIC macro program 349t
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 134–135t
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, 348t
%QSUBSTR function 130, 156, 178, 200, 348t
%QSYSFUNC function 141–147, 141f, 178, 199–200, 280, 348t
%QTRIM function 178
quotation marks 24, 41–42, 186–187
%QUOTE function 136t, 181
quoting functions
about 178
applying 182–189
commonly used 180–181
how they work 181–182
macro 135–137, 136–137t
need for 179–180
quoting versions, using of macro character functions
and autocall macro programs 198–200
%QUUPCASE function 130, 178, 265–266

R
REP16K macro program 220–223, 349t
REPGRP macro variable 28, 31, 33, 34
REPMONTH parameter 172–174
REPORT macro program
about 353t
enhancing 338
executing 328–338
in PUBLISHERSALES macro program 348t
in PUBLISHREPORT macro program 347t
refining and generalizing macro programs 326–328
in REP16K macro program 349t
REPORT procedure
CALL EXECUTE routine 220–223, 223–226
preventing misinterpretation of special characters 191–193
in RTF_START macro program 351t
standardizing RTF output 266–269, 267–269
using %IF-%THEN-%ELSE statement to modify
and select statements 162–165
REPORTA macro program 324–325, 353t
REPORTB macro program 324–325, 353t
REPORTC macro program 353t
REPORTS macro program 160–162, 347t
REPORTTITLE macro variable 185–186
REPTITLE macro program 39–40, 350t
REPTYPE parameter 160–162, 162–165
REPVAR macro variable 39–40
RESOLVE function
about 202t, 227–231
ATTRIB statement 228
in error checking 276t
FREQ procedure 229
in GETSALENAME macro program 350t
LENGTH statement 228
obtaining macro variable values with 228–229
using to call macro programs within DATA steps 230–231
RESPONSE parameter 148–150, 346t
RETAIN statement 352t
%RETURN statement 154t
RTF output, standardizing 266–269
RTF_END macro program 267–269, 351t
RTF_START macro program 267–269, 351t
RUN statement 26, 35

S
SALES macro program 343t
SAS catalogs, identifying autocall libraries stored in 252
SAS Component Language application 12t
SAS language
functions 18–19, 142–143
macro language vs. 277t
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, 68t, 69t,
70t, 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 12t
SAS/GRAPH application 12t
SASMSTORE option 254–255
%SSCAN function
  about 130, 130–131t, 156, 346t
  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 70t, 256, 258
SELECT statement 98, 232–241, 310–316
SELECTTITLES macro program 299–301, 352t
semicolon, preventing interpretation of as a SAS statement terminator 182–183
SENTENCE macro variable 54
SERROR option 276t, 279t, 281–298
SET statement 284–298
SGPANEL procedure 343t
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 347t
  in PROFITCHART macro program 345t
  in REPORTB macro program 353t
  TITLE statement 16–18
source code 257–258
SOURCE option
  creating stored compiled macro programs 255–256
%/option(s)> element 70t
saving and retrieving source code of stored compiled macro programs 257–258
special characters
  about 177–178
  category of errors 277t
  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, 348t, 350t
  in AUTHORREPORT macro program 352t
  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 353t
  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 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 238t
SQLOBS macro variable 238t, 239–241
SQLOOPS macro variable 238
SQLRC macro variable 238
SQLXMSG macro variable 238, 239
SQLXRC macro variable 238, 239
SRC option 70
standardizing RTF output 266–269
STANDARDOPTS macro program 343
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
STMT option 69
STOP parameter 80–82, 83–86, 86–89, 326–328
STORE option 70, 257–258
stored compiled facility 248
stored compiled macro programs 257–259
%STR function
about 136t, 178, 180–181, 348
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
STYLEREPOR'T parameter 191–193
%SUBSTR function 130, 131, 131t, 156, 183–184, 200, 346t, 348
SUMSOFTPROG macro variable 18
SUMYEARS macro program 169–170, 347
%SUPERQ function 136t, 137t, 178, 180–181, 188–189, 265–266, 348
SURVLIB macro variable 59
%SYMBOBL macro variable 214–216
symbol tables, specifying 214–216
SYMBOLGEN option
about 344t
creating macro programs 321–325, 328
debugging macro programming 278, 279t, 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, 152t, 346t
SYMEXIST tool 202t
%SYMEXLIST function 137t, 280
SYMGET function
about 202–209, 202t, 349t
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
%SYMGLOBAL function 137t, 280, 346t
%SYMLOCAL function 137t, 202t, 280, 346t
%SYMLOCAL macro variable 214–216
SYMPUT call routine 209–216
SYMPUTX call routine 16–18, 209–216
syntax error 274
%SYSCALL statement 152t
SYSDATE macro variable 49t, 131
SYSDATE9 macro variable 49t, 97–101
SYSDAY macro variable 39–40, 49t, 97–101, 125, 343t
SYSDEN macro variable 49t
SYSERR macro variable 49t
SYSERRORTEXT macro variable 49t, 301–303, 352t
%SYSEVALF function
about 133–135, 154, 155–156, 346t
in error checking 277t
error tracing 285–289
preventing interpretation of mnemonic operators 187–188
using 144–145
%SYSEXEC statement 153t
SYSFILRC macro variable 49t
%SYSFUNC function
  about 141–142, 346t
  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 343t
  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, 141f, 149–150
%SYSGET function 141f
SYSLAST macro variable 49t
SYSLIBRC macro variable 49t
%SYSLPUT statement 153t
%SYSMACDELETE statement 73
SYSMACRONAME macro variable 49t
SYSPBUFF macro variable 89–93
%SYSPROD function 141f
%SYSRPUT statement 153t
system information, displaying 12–13
system option settings category of errors 276t
SYSTIME macro variable 49t
SYSVER macro variable 49t, 97–101
SYSWARNINGTEXT macro variable 49t, 301–303, 352t

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
  macro variables as 38–39
  submitting %PUT statement to display 45–46
  <text> element 70t
timing, of CALL EXECUTE routine 217–218, 219–221
TITLE statement
  about 3, 133, 343t, 344t, 346t
  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 348t
  passing information between program steps 16–18
  in REPTITLE macro program 350t
  resolving macro variables enclosed in quotation
  marks 41–42
  in RTF_END macro program 351t
  in STANDARDOPTS macro program 343t
  in STATSECTION macro program 349t
  using %UNQUOTE function 198
Index 367

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

U
UNIX 20, 252–254
unmasking text 198
%UNQUOTE function 137f, 178, 198, 348f
%UPCASE function 130, 131f, 132–133, 148–150
%UPCASE macro program 346f
uppercase, converting macro variable values to with
%UPCASE 132–133
_USER_ option 43f
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, 347f
%VERIFY function 148–150, 148f, 346f

W
WARNING: option 43f, 44
WHERE statement
about 133, 343f, 344f, 346f
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 352f
WHEREVAR macro variable 64
WHSTMT macro program 283–298, 284–298, 352f
%WINDOW statement 154f
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 152f
- operator 155f
/ operator 155f
* operator 155f
** operator 155f
^ operator 155f
+ operator 155f
~ operator 155f
|| (concatenation) operators 154
>= operator 155f
& (ampersand) 7, 24, 60–61, 64–65, 136t, 137f, 180, 232
& (AND) operator 155f
: (colon) 44, 174, 232
- (dash) 44
= (EQ) operator 155f
= (equal sign) 82
>(GE) operator 155f
<table>
<thead>
<tr>
<th>Operator</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~=</code> (GT)</td>
<td>155t</td>
</tr>
<tr>
<td><code>#</code> (IN)</td>
<td>155t</td>
</tr>
<tr>
<td><code>&lt;=</code> (LE)</td>
<td>155t</td>
</tr>
<tr>
<td><code>&lt;</code> (LT)</td>
<td>155t</td>
</tr>
<tr>
<td><code>^=</code> (NE)</td>
<td>155t</td>
</tr>
<tr>
<td><code>~</code> (NOT)</td>
<td>154, 155t</td>
</tr>
<tr>
<td>`</td>
<td>` (OR)</td>
</tr>
<tr>
<td><code>%</code> (percent sign)</td>
<td>7, 24, 136t, 137t, 174, 180, 186–187</td>
</tr>
<tr>
<td><code>.</code> (periods)</td>
<td>55</td>
</tr>
</tbody>
</table>
About The Author


Learn more about this author by visiting her author page at: [http://support.sas.com/burlew](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.