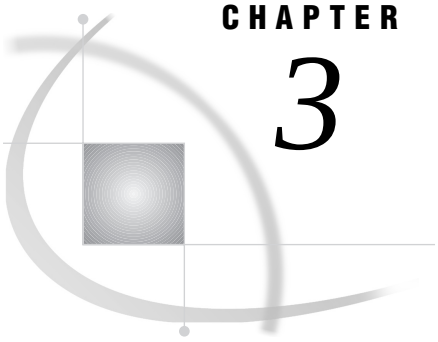

CHAPTER

3

Macro Variables

Introduction

Macro variables are tools that enable you to dynamically modify the text in a SAS program through symbolic substitution. You can assign large or small amounts of text to macro variables, and after that, you can use that text by simply referencing the variable that contains it.

Macro variable values have a maximum length of 32K characters. The length of a macro variable is determined by the text assigned to it instead of an explicit length declaration. So its length varies with each value it contains. Macro variables contain only character data. However, the macro facility has features that allow a variable to be evaluated as a number when it contains a value that can be interpreted as a number. The value of a macro variable remains constant until it is explicitly changed. Macro variables are independent of SAS data set variables.

Macro variables defined by macro programmers are called *user-defined macro variables*. Those defined by the SAS System are called *automatic macro variables*. You can define and use macro variables anywhere in SAS programs, except within data lines.

When a macro variable is defined, the macro processor adds it to one of the program's macro variable symbol tables. When a macro variable is defined in a statement that is outside a macro definition (called *open code*) or when the variable is created automatically by the SAS System (except SYSPBUFF), the variable is held in the global symbol table, which SAS creates at the beginning of a SAS session. When a macro variable is defined within a macro and is not explicitly defined as global, the variable is typically held in the macro's local symbol table, which SAS creates when the macro starts executing. For more information about symbol tables, see Chapter 2, "SAS Programs and Macro Processing" and Chapter 5, "Scope of Macro Variables."

When it is in the global symbol table, a macro variable exists for the remainder of the current SAS session. A variable in the global symbol table is called a *global macro variable*. It has global scope because its value is available to any part of the SAS session.

When it is in a local symbol table, a macro variable exists only during execution of the macro in which it is defined. A variable in a local symbol table is called a *local macro variable*. It has local scope because its value is available only until the macro stops executing. Chapter 2 contains figures that illustrate a program with a global and a local symbol table.

You can use the %PUT statement to view all macro variables available in a current SAS session. See %PUT in Chapter 13, "Macro Language Dictionary," and also in Chapter 10, "Macro Facility Error Messages and Debugging."

Macro Variables Defined by the SAS System

When you invoke SAS, the macro processor creates automatic macro variables that supply information related to the SAS session. Automatic variables are global except SYSPBUFF, which is local.

To use an automatic macro variable, reference it with an ampersand followed by the macro variable name (for example, &SYSJOBID). This FOOTNOTE statement contains references to the automatic macro variables SYSDAY and SYSDATE:

```
footnote "Report for &sysday, &sysdate";
```

If the current SAS session is invoked on December 17, 1996, macro variable resolution causes SAS to receive this statement:

```
FOOTNOTE "Report for Tuesday, 17DEC96";
```

Automatic macro variables are often useful in conditional logic such as a %IF statement with actions determined by the value that is returned. %IF is described in Chapter 13.

You can assign values to automatic macro variables that have read/write status. However, you cannot assign a value to an automatic macro variable that has read-only status. Table 3.1 on page 18 lists the automatic macro variables that are supplied by base SAS software and their read/write status. They are described in Chapter 13.

Use %PUT _AUTOMATIC_ to view all available automatic macro variables.

There are also system-specific macro variables that are created only on a particular platform. These are documented in the host companion, and common ones are listed in Chapter 11. Other SAS software products also provide macro variables, which are described in the documentation for the product that uses them.

Table 3.1 Automatic Macro Variables by Category

Status	Variable	Contains
Read/Write	SYSBUFFER	unmatched text from %INPUT
	SYSCC	the current condition code that SAS returns to your operating environment (the operating environment condition code)
	SYSCMD	last unrecognized command from the command line of a macro window
	SYSDEVIC	name of current graphics device

Status	Variable	Contains
	SYSDMG	return code that reflects an action taken on a damaged data set
	SYSDSN	name of most recent SAS data set in two fields
	SYSFILRC	return code set by the FILENAME statement
	SYSLAST	name of most recent SAS data set in one field
	SYSLCKRC	return code set by the LOCK statement
	SYSLIBRC	return code set by the LIBNAME statement
	SYSMSG	message for display in macro window
	SYSPARM	value specified with the SYSPARM= system option
	SYSPBUFF	text of macro parameter values
	SYSRC	various system-related return codes
Read-Only	SYSCHARWIDTH	the character width value
	SYSDATE	the character value representing the date a SAS job or session began executing (two-digit year)
	SYSDATE9	the character value representing the date a SAS job or session began executing (four-digit year)
	SYSDAY	day of week SAS job or session began executing
	SYSENV	foreground or background indicator
	SYSERR	return code set by SAS procedures and the DATA step
	SYSINDEX	number of macros that have begun execution during this session
	SYSINFO	return code information
	SYSJOBID	name of current batch job or userid (varies by host environment)
	SYSMENV	current macro execution environment
	SYSPROCESSID	the process id of the current SAS process
	SYSPROCESSNAME	the process name of the current SAS process
	SYSSCP	the abbreviation of an operating system
	SYSSCPL	the name of an operating system
	SYSSITE	the number assigned to your site
	SYSSTARTID	the id generated from the last STARTSAS statement
	SYSSTARTNAME	the process name generated from the last STARTSAS statement
	SYSTIME	the character value of the time a SAS job or session began executing
	SYSUSERID	the userid or login of the current SAS process

Status	Variable	Contains
	SYSVER	release or version number of SAS software executing
	SYSVLONG	release number and maintenance level of SAS software

Macro Variables Defined by Users

You can create your own macro variables, change their values, and define their scope. You can define a macro variable within a macro, and you can also explicitly define it as a global variable, by defining it with the %GLOBAL statement. Macro variable names must start with a letter or an underscore and can be followed by letters or digits. You can assign any name to a macro variable as long as the name is not a reserved word. The prefixes AF, DMS, SQL, and SYS are not recommended because they are frequently used in SAS software for automatic macro variables. Thus, using one of these prefixes can cause a name conflict with an automatic macro variable. For a complete list of reserved words in the macro language, see Appendix 1, "Reserved Words in the Macro Facility." If you assign a macro variable name that is not valid, an error message is printed in the SAS log.

You can use %PUT _ALL_ to view all user-created macro variables. See %PUT in Chapter 13.

Creating Macro Variables and Assigning Values

The simplest way to create and assign a value to a macro variable is to use the macro program statement %LET, as in

```
%let dsname=Newdata;
```

DSNAME is the name of the macro variable. **Newdata** is the value of the macro variable DSNAME. The value of a macro variable is simply a string of characters. The characters can include any letters, numbers, or printable symbols found on your keyboard, and blanks between characters. The case of letters is preserved in a macro variable value. Some characters, such as unmatched quotation marks, require special treatment, which is described later.

If a macro variable already exists, a value assigned to it replaces its current value. If a macro variable or its value contains macro triggers (% or &), the trigger is evaluated before the value is assigned. In the following example, **&name** is resolved to **Cary** and then it is assigned as the value of **city** in the following statements:

```
%let name=Cary;
%let city=&name;
```

Generally, the macro processor treats alphabetic characters, digits, and symbols (except & and %) as characters. It can also treat & and % as characters using a special treatment, which is described later. It does not make a distinction between character and numeric values as the rest of the SAS System does. (However, the %EVAL and %SYSEVALF functions can evaluate macro variables as integers or floating point numbers. See "Evaluation Functions" in Chapter 12, "Macro Language Elements.")

Macro variable values can represent text to be generated by the macro processor or text to be used by the macro processor. Values can range in length from 0 bytes to 32K. If you omit the value argument, the value is null (0 characters). By default, leading and trailing blanks are not stored with the value.

In addition to the %LET statement, other features of the macro language that create macro variables are

- iterative %DO statement
- %GLOBAL statement
- %INPUT statement
- INTO clause of the SELECT statement in SQL
- %LOCAL statement
- %MACRO statement
- SYMPUT routine and SYMPUTN routine in SCL
- %WINDOW statement.

Table 3.2 on page 21 describes how to assign a variety of types of values to macro variables.

Table 3.2 Types of Assignments for Macro Variable Values

To assign ...	Use...

Constant text	a character string. The following statements show several ways that the value **maple** can be assigned to macro variable STREET. In each case, the macro processor stores the five-character value **maple** as the value of STREET. The leading and trailing blanks are not stored.
	`%let street=maple;`
	
	`%let street= maple;`
	`%let street=`
	`maple;`
	Note: Quotation marks are not required. If quotation marks are used, they become part of the value.

Digits	the appropriate digits. This example creates the macro variables NUM and TOTALSTR:
	`%let num=123'`
	`%let totalstr=100+200;`
	The macro processor does not treat **123** as a number or evaluate the expression **100+200**. Instead, the macro processor treats all the digits as characters.

Arithmetic expressions	the %EVAL function, for example,
	`%let num=%eval(100+200); / * produces 300 * /`
	use the %SYSEVALF function, for example,
	`%let num=%sysevalf(100+1.597) / * produces 101.597 * /`
	For more information, see "Evaluation Functions" in Chapter 12 and details on the functions in Chapter 13.

A null value	no assignment for the value argument. For example, `%let country=;`

To assign ...	Use...
A macro variable reference	a macro variable reference, &*macro-variable*. For example, **%let street=Maple;** **%let num=123;** **%let address=&num &street Avenue;** This example shows multiple macro references that are part of a text expression. The macro processor attempts to resolve text expressions before it makes the assignment. Thus, the macro processor stores the value of macro variable ADDRESS as **123 Maple Avenue**. You can treat ampersands and percent signs as literals by using the %NRSTR function to mask the character so that the macro processor treats it as text instead of trying to interpret it as a macro call. See “Quoting Functions” in Chapter 12 and Chapter 7, “Macro Quoting,” for information.
A macro invocation	a macro call, %*macro-name*. For example, **%let status=%wait;** When the %LET statement executes, the macro processor also invokes the macro WAIT. The macro processor stores the text produced by the macro WAIT as the value of STATUS. To prevent the macro from being invoked when the %LET statement executes, use the %NRSTR function to mask the percent sign: **%let status=%nrstr(%wait);** The macro processor stores **%wait** as the value of STATUS.

To assign ...	Use...
Blanks and special characters	macro quoting function %STR or %NRSTR around the value. This action masks the blanks or special characters so that the macro processor interprets them as text. See "Quoting Functions" in Chapter 12 and Chapter 7, "Macro Quoting." For example, **%let state=%str(North Carolina);** **%let town=%str(Taylor%'s Pond);** **%let store=%nrstr(Smith&Jones);** **%let plotit=%str(** **proc plot;** **plot income*age;** **run;);** The definition of macro variable TOWN demonstrates using %STR to mask a value containing an unmatched quotation mark. "Macro Quoting Functions" in Chapter 12 and Chapter 7 discuss macro quoting functions that require unmatched quotation marks and other symbols to be marked. The definition of macro variable PLOTIT demonstrates using %STR to mask blanks and special characters (semicolons) in macro variable values. When a macro variable contains complete SAS statements, the statements are easier to read if you enter them on separate lines with indentions for statements within a DATA or PROC step. Using a macro quoting function retains the significant blanks in the macro variable value.
Value from a DATA step	the SYMPUT routine. This example puts the number of observations in a data set into a FOOTNOTE statement where AGE is greater than 20: **data _null_;** **set in.permdata end=final;** **if age>20 then n+1;** **if final then call symput('number',trim(left(n)));** **run;** **footnote "&number Observations have AGE>20";** During the last iteration of the DATA step, the SYMPUT routine creates a macro variable named NUMBER whose value is the value of N. (The SAS System also issues a numeric-to-character conversion message.) The TRIM and the LEFT functions remove the extra space characters from the DATA step variable N before its value is assigned to the macro variable NUMBER. The program generates this FOOTNOTE statement: **FOOTNOTE "Observations have AGE>20";** For a discussion of SYMPUT, including information on preventing the numeric-character message, see Chapter 13.

Using Macro Variables

After a macro variable is created, you typically use the variable by referencing it with an ampersand preceding its name (&*variable-name*), which is called a *macro variable reference.* These references perform symbolic substitutions when they resolve to their value. You can use these references anywhere in a SAS program. To resolve a macro variable reference that occurs within a literal string, enclose the string in double quotation marks. Macro variable references that are enclosed in single quotation marks are not resolved. Compare the following statements that assign a value to macro variable DSN and use it in a TITLE statement:

```
%let dsn=Newdata;
title1 "Contents of Data Set &dsn";
title2 'Contents of Data Set &dsn';
```

In the first TITLE statement, the macro processor resolves the reference by replacing &DSN with the value of macro variable DSN. In the second TITLE statement, the value for DSN does not replace &DSN. The SAS System sees the following statements:

```
TITLE1 "Contents of Data Set Newdata";
TITLE2 'Contents of Data Set &dsn';
```

You can refer to a macro variable as many times as you need to in a SAS program. The value remains constant until you change it. For example, this program refers to macro variable DSN twice:

```
%let dsn=Newdata;
data temp;
   set &dsn;
   if age>=20;
run;

proc print;
   title "Subset of Data Set &dsn";
run;
```

Each time the reference &DSN appears, the macro processor replaces it with **Newdata**. Thus, the SAS System sees these statements:

```
DATA TEMP;
      SET NEWDATA;
      IF AGE>=20;
   RUN;

   PROC PRINT;
      TITLE "Subset of Data Set NewData";
   RUN;
```

Note: If you reference a macro variable that does not exist, a warning message is printed in the SAS log. For example, if macro variable JERRY is misspelled as JERY, the following produces an unexpected result:

```
%let jerry=student;
data temp;
  x="produced by &jery";
run;
```

This produces the following message:

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

△

Combining Macro Variable References with Text

It is often useful to place a macro variable reference next to leading or trailing text (for example, DATA=PERSNL&YR.EMPLOYES, where &YR contains two characters for a year), or to reference adjacent variables (for example, &MONTH&YR). This allows you to reuse the same text in several places or to reuse a program because you can change values for each use.

To reuse the same text in several places, you can write a program with macro variable references representing the common elements. You can change all the locations with a single %LET statement, as shown:

```
%let name=sales;
   data new&name;
      set save.&name;
      more SAS statements
      if units>100;
   run;
```

After macro variable resolution, the SAS System sees these statements:

```
DATA NEWSALES;
      SET SAVE.SALES;
      more SAS statements
      IF UNITS>100;
   RUN;
```

Notice that macro variable references do not require the concatenation operator as the DATA step does. The SAS System forms the resulting words automatically.

Delimiting Macro Variable Names within Text

Sometimes when you use a macro variable reference as a prefix, the reference does not resolve as you expect if you simply concatenate it. Instead, you may need to delimit the reference by adding a period to the end of it.

A period immediately following a macro variable reference acts as a delimiter; that is, a period at the end of a reference forces the macro processor to recognize the end of the reference. The period does not appear in the resulting text.

Continuing with the example above, suppose that you need another DATA step that uses the names SALES1, SALES2, and INSALES.TEMP. You might add the following step to the program:

```
/*  first attempt to add suffixes--incorrect  */
data &name1 &name2;
   set in&name.temp;
run;
```

After macro variable resolution, the SAS System sees these statements:

```
DATA &NAME1 &NAME2;
   SET INSALESTEMP;
RUN;
```

None of the macro variable references have resolved as you intended. The macro processor issues warning messages, and the SAS System issues syntax error messages. Why?

Because NAME1 and NAME2 are valid SAS names, the macro processor searches for those macro variables rather than for NAME, and the references pass into the DATA statement without resolution.

In a macro variable reference, the word scanner recognizes that a macro variable name has ended when it encounters a character that is not allowed in a SAS name. However, you can use a period (.) as a delimiter for a macro variable reference. For example, to cause the macro processor to recognize the end of the word NAME in this example, use a period as a delimiter between &NAME and the suffix:

```
/*  correct version  */
data &name.1 &name.2;
```

The SAS System now sees this statement:

```
DATA SALES1 SALES2;
```

Creating a Period to Follow Resolved Text

Sometimes you need a period to follow the text resolved by the macro processor. For example, a two-level data set name needs to include a period between the libref and data set name.

When the character following a macro variable reference is a period, use two periods. The first is the delimiter for the macro reference, and the second is part of the text. For example,

```
set in&name..temp;
```

After macro variable resolution, the SAS System sees this statement:

```
SET INSALES.TEMP;
```

You can end any macro variable reference with a delimiter, but the delimiter is necessary only if the characters that follow can be part of a SAS name. For example, both of these TITLE statements are correct:

```
title "&name.--a report";
   title "&name--a report";
```

They produce:

```
TITLE "sales--a report";
```

Displaying Macro Variable Values

The simplest way to display macro variable values is to use the %PUT statement, which writes text to the SAS log. For example, the statements

```
%let a=first;
%let b=macro variable;
%put &a ***&b***;
```

write

```
first ***macro variable***
```

You can also use a %PUT statement to view available macro variables. %PUT provides several options that allow you to view individual categories of macro variables. %PUT is described in Chapter 13.

The system option SYMBOLGEN displays the resolution of macro variables. For this example, assume that macro variables PROC and DSET have the values GPLOT and SASUSER.HOUSES, respectively.

```
options symbolgen;
%let title "%upcase(&proc) of %upcase(&dset)";
```

The SYMBOLGEN option prints to the log:

```
SYMBOLGEN:  Macro variable PROC resolves to gplot
SYMBOLGEN:  Macro variable DATA resolves to sasuser.houses
```

For more information on debugging macro programs, see Chapter 10, “Macro Facility Error Messages and Debugging.”

Referencing Macro Variables Indirectly

The macro variable references shown so far have been direct macro references that begin with one ampersand: &*name*. However, it is also useful to be able to indirectly reference macro variables that belong to a series so that the name is determined when the macro variable reference resolves. The macro facility provides indirect macro variable referencing, which allows you to use an expression (for example, CITY&N) to generate a reference to one of a series of macro variables. For example, you could use the value of macro variable N to reference a variable in the series of macro variables named CITY1 to CITY20. If N has the value 8, the reference would be to CITY8. If the value of N is 3, the reference would be to CITY3.

Although for this example the type of reference you want is CITY&N, the following example will not produce the results that you expect, which is the value of &N appended to CITY:

```
%put &city&n;  /* incorrect */
```

This produces a warning message saying that there is no macro variable CITY because the macro facility has tried to resolve &CITY and then &N and concatenate those values.

When you use an indirect macro variable reference, you must force the macro processor to scan the macro variable reference more than once and resolve the desired reference on the second, or later, scan. To force the macro processor to rescan a macro variable reference, you use more than one ampersand in the macro variable reference. When the macro processor encounters multiple ampersands, its basic action is to resolve two ampersands to one ampersand. For example, to append the value of &N to CITY and then reference the appropriate variable name, you use:

```
%put &&city&n;  /* correct */
```

Assuming that &N contains 6, when the macro processor receives this statement, it performs the following steps:

1. resolves && to &
2. passes CITY as text
3. resolves &N into 6
4. returns to the beginning of the macro variable reference, &CITY6, starts resolving from the beginning again, and prints the value of CITY6.

Generating a Series of Macro Variable References with a Single Macro Call

Using indirect macro variable references, you can generate a series of references with a single macro call by using an iterative %DO loop. The following example assumes that the macro variables CITY1 through CITY10 contain the respective values Cary, New York, Chicago, Los Angeles, Austin, Boston, Orlando, Dallas, Knoxville, and Asheville:

```
%macro listthem;
   %do n=1 %to 10;&city&n
   %end;
%mend listthem;

%put %listthem;
```

This program writes the following to the SAS log:

```
Cary    New York    Chicago    Los Angeles    Austin    Boston
   Orlando      Dallas      Knoxville      Asheville
```

Using More Than Two Ampersands

You can use any number of ampersands in an indirect macro variable reference, although using more than three is rare. Regardless of how many ampersands are used in this type of reference, the macro processor performs the following steps to resolve the reference. For example,

```
%let var=city;
%let n=6;
%put &&&var&n;
```

1. It resolves the entire reference from left-to-right. If a pair of ampersands (&&) is encountered, the pair is resolved to a single ampersand, then the next part of the reference is processed. In this example, &&&VAR&N becomes &CITY6.
2. It returns to the beginning of the preliminary result and starts resolving again from left-to-right. When all ampersands have been fully processed, the resolution is complete. In this example, &CITY6 resolves to Boston, and the resolution process is finished.

In some cases, using indirect macro references with triple ampersands increases the efficiency of the macro processor. For more information see Chapter 11, “Writing Efficient and Portable Macros.”

Manipulating Macro Variable Values with Macro Functions

When you define macro variables, you can include macro functions in the expressions to manipulate the value of the variable before the value is stored. For example, you can use functions that scan other values, evaluate arithmetic and logical expressions, and remove the significance of special characters such as unmatched quotation marks.

To scan for words in macro variable values, use the %SCAN function. For example,

```
%let address=123 maple avenue;
%let frstword=%scan(&address,1);
```

The first %LET statement assigns the string **123 maple avenue** to macro variable ADDRESS. The second %LET statement uses the %SCAN function to search the source (first argument) and retrieve the first word (second argument). Because the macro processor executes the %SCAN function before it stores the value, the value of FRSTWORD is the string **123**. (The %SCAN function is discussed in Chapter 13.)

For more information about macro functions, see “Macro Functions” in Chapter 12.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Macro Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 310 pages.

SAS Macro Language: Reference, Version 8

1-58025-522-1

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

1st printing, October 1999