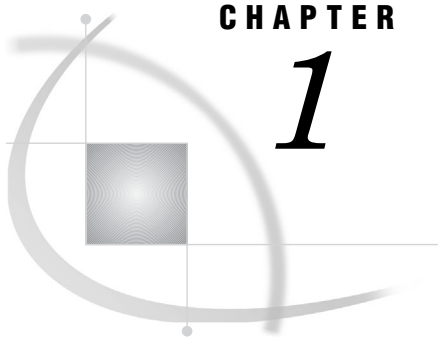


CHAPTER

1

Introduction to the Macro Facility

Getting Started with the Macro Facility

This is the macro facility language reference for SAS. It is a reference for the SAS macro language processor and defines the SAS macro language elements. This section introduces the SAS macro facility using simple examples and explanation.

The *macro facility* is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do common tasks. The macro facility enables you to assign a name to character strings or groups of SAS programming statements. From that point on, you can work with the names rather than with the text itself.

When you use a macro facility name in a SAS program or from a command prompt, the macro facility generates SAS statements and commands as needed. The rest of SAS receives those statements and uses them in the same way it uses the ones you enter in the standard manner.

The macro facility has two components:

the *macro processor*	is the portion of SAS that does the work.
the *macro language*	is the syntax that you use to communicate with the macro processor.

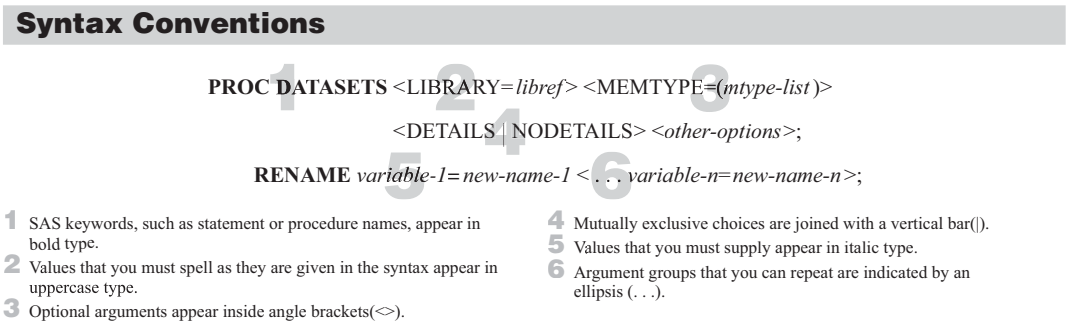
When SAS compiles program text, two delimiters trigger macro processor activity:

&*name*	refers to a macro variable. “Replacing Text Strings Using Macro Variables” on page 4 explains how to create a macro variable. The form &*name* is called a macro variable reference.
%name	refers to a macro. “Generating SAS Code Using Macros” on page 5 explains how to create a macro. The form *%name* is called a macro call.

The text substitution produced by the macro processor is completed *before* the program text is compiled and executed. The macro facility uses statements and functions that resemble those that you use in the DATA step. An important difference, however, is that macro language elements can only trigger text substitution and are not present during program or command execution.

Note: Three SAS statements begin with a % that are not part of the macro facility. These elements are the %INCLUDE, %LIST, and %RUN statements. These statements are documented in your Base SAS documentation. △

The following graphic explains the syntax used in this document:



Replacing Text Strings Using Macro Variables

Macro variables are an efficient way of replacing text strings in SAS code. The simplest way to define a macro variable is to use the %LET statement to assign the macro variable a name (subject to standard SAS naming conventions), and a value. Here is a simple example:

```
%let city=New Orleans;
```

Now you can use the macro variable CITY in SAS statements where you'd like the text **`New Orleans`** to appear. You refer to the variable by preceding the variable name with an ampersand (&), as in the following TITLE statement:

```
title "Data for &city";
```

The macro processor resolves the reference to the macro variable CITY, and the statement becomes

```
title "Data for New Orleans";
```

A macro variable can be defined within a macro definition or within a statement that is outside a macro definition (called *open code*).

Note: The title is enclosed in double quotation marks. In quoted strings in open code, the macro processor resolves macro variable references within double quotation marks but not within single quotation marks. △

A %LET statement in open code (outside a macro definition) creates a global macro variable that is available for use anywhere in your SAS code during the SAS session in which the variable was created. There are also *local* macro variables, which are available for use only inside the macro definition where they are created. See Chapter

5, "Scopes of Macro Variables," on page 41 for more information on global and local macro variables.

Macro variables are not subject to the same length limits as SAS data set variables. However, if the value you want to assign to a macro variable contains certain special characters (for example, semicolons, quotation marks, ampersands, and percent signs) or mnemonics (for example, AND, OR, or LT), you must use a macro quoting function to mask the special characters. Otherwise, the special character or mnemonic might be misinterpreted by the macro processor. See Chapter 7, "Macro Quoting," on page 75 for more information on macro quoting.

While macro variables are useful for simple text substitution, they cannot perform conditional operations, DO loops, and other more complex tasks. For this kind of work, you must define a macro.

Generating SAS Code Using Macros

Macros enable you to substitute text in a program and to do many other things. A SAS program can contain any number of macros, and you can invoke a macro any number of times in a single program.

To help you learn how to define your own macros, this section presents a few examples you can model your own macros after. Each of these examples is fairly simple; by mixing and matching the various techniques, you can create advanced, flexible macros that are capable of performing complex tasks.

Each macro you define has a distinct name, which is subject to the standard SAS naming conventions. (See the Base SAS language documentation for more information on SAS naming conventions.) A macro definition is placed between a %MACRO statement and a %MEND (macro end) statement, as follows:

%MACRO *macro-name*;
    macro definition

%MEND *macro-name*;

The *macro-name* specified in the %MEND statement must match the *macro-name* specified in the %MACRO statement.

Note: While specifying the *macro-name* in the %MEND statement is not required, it is recommended. It makes matching %MACRO and %MEND statements while debugging easier. △

Here is a simple macro definition:

```
%macro dsn;
   Newdata
%mend dsn;
```

This macro is named DSN. **`Newdata`** is the text of the macro. A string inside a macro is called *constant text* or *model text* because it is the model, or pattern, for the text that becomes part of your SAS program.

To call (or *invoke*) a macro, precede the name of the macro with a percent sign (%), as follows:

%macro-name

Although the call to the macro looks somewhat like a SAS statement, it does not have to end in a semicolon.

For example, here is how you might call the DSN macro:

```
title "Display of Data Set %dsn";
```

The macro processor executes the macro DSN, which substitutes the constant text in the macro into the TITLE statement. Thus, the TITLE statement becomes

```
title "Display of Data Set Newdata";
```

Note: The title is enclosed in double quotation marks. In quoted strings in open code, the macro processor resolves macro invocations within double quotation marks but not within single quotation marks. △

The macro DSN is exactly the same as coding the following:

```
%let dsn=Newdata;

title "Display of Data Set &dsn";
```

The result is still

```
title "Display of Data Set Newdata";
```

So, in this case, the macro approach does not have any advantages over the macro variable approach. However, DSN is an extremely simple macro. As you will see in later examples, macros can do much more than the macro DSN does.

Inserting Comments in Macros

All code benefits from thorough commenting, and macro code is no exception. There are two forms you can use to add comments to your macro code.

The first form is the same as comments in SAS code, beginning with `/*` and ending with `*/`. The second form begins with a `%*` and ends with a `;`. Here is a program that uses both types of comments:

```
%macro comment;
/* Here is the type of comment used in other SAS code. */
   %let myvar=abc;

%* Here is a macro-type comment.;
   %let myvar2=xyz;

%mend comment;
```

You can use whichever type comment you prefer in your macro code, or use both types as in the previous example.

The asterisk-style comment (* *commentary* ;)used in SAS code is not recommended within a macro definition. While the asterisk-style will comment constant text appropriately, it will execute any macro statements contained within the comment.

Macro Definition Containing Several SAS Statements

You can create macros that contain entire sections of a SAS program:

```
%macro plot;
   proc plot;
      plot income*age;
   run;
```

```
%mend plot;
```

Later in the program you can invoke the macro as follows:

```
data temp;
   set in.permdata;
   if age>=20;
run;

%plot

proc print;
run;
```

Executing these statements produces the following program:

```
data temp;
   set in.permdata;
   if age>=20;
run;

proc plot;
   plot income*age;
run;

proc print;
run;
```

Passing Information into a Macro Using Parameters

A macro variable defined in parentheses in a %MACRO statement is a *macro parameter*. Macro parameters enable you to pass information into a macro. Here is a simple example:

```
%macro plot(yvar= ,xvar= );
   proc plot;
      plot &yvar*&xvar;
   run;
%mend plot;
```

You invoke the macro by providing values for the parameters, as follows:

```
%plot(yvar=income,xvar=age)

%plot(yvar=income,xvar=yrs_educ)
```

When the macro executes, the macro processor matches the values specified in the macro call to the parameters in the macro definition. (This type of parameter is called a *keyword parameter*.)

Macro execution produces the following code:

```
proc plot;
   plot income*age;
run;

proc plot;
   plot income*yrs_educ;
run;
```

Using parameters has several advantages. First, you can write fewer %LET statements. Second, using parameters ensures that the variables never interfere with parts of your program outside the macro. Macro parameters are an example of *local* macro variables, which exist only during the execution of the macro in which they are defined.

Conditionally Generating SAS Code

By using the %IF-%THEN-%ELSE macro statements, you can conditionally generate SAS code with a macro. Here is an example:

```
%macro whatstep(info=,mydata=);
   %if &info=print %then
      %do;
         proc print data=&mydata;
         run;
      %end;

   %else %if &info=report %then
      %do;
         options nodate nonumber ps=18 ls=70 fmtsearch=(sasuser);
      proc report data=&mydata nowd;
         column manager dept sales;
         where sector='se';
         format manager $mgrfmt. dept $deptfmt. sales dollar11.2;
         title 'Sales for the Southeast Sector';
      run;
   %end;
%mend whatstep;
```

In this example, the macro WHATSTEP uses keyword parameters, which are set to default null values. When you call a macro that uses keyword parameters, specify the parameter name followed by an equal sign and the value you want to assign the parameter. Here, the macro WHATSTEP is called with INFO set to **print** and MYDATA set to **grocery**:

```
%whatstep(info=print,mydata=grocery)
```

This produces the following statements:

```
proc print data=grocery;
run;
```

Because values in the macro processor are case sensitive, the previous program does not work if you specify **PRINT** instead of **print**. To make your macro more robust, use the %UPCASE macro function. For more information on this function, refer to Chapter 13, "Macro Language Dictionary," on page 163.

For more information on macro definitions and macro parameters, see %MACRO and %MEND in Chapter 13, "Macro Language Dictionary," on page 163.

More Advanced Macro Techniques

After mastering the basic techniques previously discussed, you might want to learn some more advanced macro techniques.

Generating Repetitive Pieces of Text Using %DO Loops

"Conditionally Generating SAS Code" on page 8 presents a %DO-%END group of statements to conditionally execute several SAS statements. To generate repetitive pieces of text, use an iterative %DO loop. For example, the following macro, NAMES, uses an iterative %DO loop to create a series of names to be used in a DATA statement:

```
%macro names(name= ,number= );
   %do n=1 %to &number;
      &name&n
   %end;
%mend names;
```

The macro NAMES creates a series of names by concatenating the value of the parameter NAME and the value of the macro variable N. You supply the stopping value for N as the value of the parameter NUMBER, as in the following DATA statement:

```
data %names(name=dsn,number=5);
```

Submitting this statement produces the following complete DATA statement:

```
data dsn1 dsn2 dsn3 dsn4 dsn5;
```

Note: You can also execute a %DO loop conditionally with %DO %WHILE and %DO %UNTIL statements. See Chapter 13, "Macro Language Dictionary," on page 163 for details about these statements. △

Generating a Suffix for a Macro Variable Reference

Suppose that, when you generate a numbered series of names, you always want to put the letter X between the prefix and the number. The macro NAMESX inserts an X after the prefix you supply:

```
%macro namesx(name=,number=);
   %do n=1 %to &number;
      &name.x&n
   %end;
%mend namesx;
```

The period is a delimiter at the end of the reference &NAME. The macro processor uses the delimiter to distinguish the reference &NAME followed by the letter X from the reference &NAMEX. Here is an example of calling the macro NAMESX in a DATA statement:

```
data %namesx(name=dsn,number=3);
```

Submitting this statement produces the following statement:

```
data dsnx1 dsnx2 dsnx3;
```

See Chapter 3, "Macro Variables," on page 19 for more information about using a period as a delimiter in a macro variable reference.

Other Features of the Macro Language

Although subsequent sections go into far more detail on the various elements of the macro language, this section highlights some of the possibilities, with pointers to more information.

macro statements
: This section has illustrated only a few of the macro statements, such as %MACRO and %IF-%THEN. Many other macro statements exist, some of which are valid in open code, while others are valid only in macro definitions. For a complete list of macro statements, refer to "Macro Statements" on page 147.

macro functions
: Macro functions are functions defined by the macro facility. They process one or more arguments and produce a result. For example, the %SUBSTR function creates a substring of another string, while the %UPCASE function converts characters to uppercase. A special category of macro functions, the macro quoting functions, mask special characters so they are not misinterpreted by the macro processor.

  There are two special macro functions, %SYSFUNC and %QSYSFUNC, that provide access to SAS language functions or user-written functions generated with SAS/TOOLKIT. You can use %SYSFUNC and %QSYSFUNC with new functions in Base SAS software to obtain the values of SAS host, base, or graphics options. These functions also enable you to open and close SAS data sets, test data set attributes, or read and write to external files. Another special function is %SYSEVALF, which enables your macros to perform floating-point arithmetic.

  For a list of macro functions, refer to "Macro Functions" on page 149. For a discussion of the macro quoting functions, refer to Chapter 7, "Macro Quoting," on page 75. For the syntax of calling selected Base SAS functions with %SYSFUNC, refer to Appendix 3, "Syntax for Selected Functions Used with the %SYSFUNC Function," on page 321.

autocall macros
: Autocall macros are macros defined by SAS that perform common tasks, such as trimming leading or trailing blanks from a macro variable's value or returning the data type of a value. For a list of autocall macros, refer to "Selected Autocall Macros Provided with SAS Software" on page 157.

automatic macro variables
: Automatic macro variables are macro variables created by the macro processor. For example, SYSDATE contains the date SAS is invoked. See Chapter 12, "Macro Language Elements," on page 147 for a list of automatic macro variables, and Chapter 13, "Macro Language Dictionary," on page 163 for a description of these automatic macro variables.

macro facility interfaces
: Interfaces with the macro facility provide a dynamic connection between the macro facility and other parts of SAS, such as the DATA step, SCL code, the SQL procedure, and SAS/CONNECT software. For example, you can create macro variables based on values within the DATA step using CALL SYMPUT and retrieve the value of a macro variable stored on a remote host using the %SYSRPUT macro statement. For more information on these interfaces, refer to Chapter 8, "Interfaces with the Macro Facility," on page 95.