CHAPTER

7

Macro Quoting

Overview of Macro Quoting

The macro language is a character-based language. Even variables that appear to be numeric are generally treated as character variables (except during expression evaluation). Therefore, the macro processor enables you to generate all sorts of special characters as text; but because the macro language is composed of some of the same special characters, an ambiguity often arises: should the macro processor interpret a particular special character (for example, a semicolon or % sign) or a mnemonic (for example, GE or AND) as text or as a symbol in the macro language? Macro quoting functions resolve these ambiguities by masking the significance of special characters so the macro processor does not misinterpret them.

The following special characters and mnemonics may require masking when they appear in text strings:

blank	)	=	LT
;	(	|	GE
¬	+	AND	GT
^	—	OR	%
~	*	NOT	&
, (comma)	/	EQ	
'	<	NE	
"	>	LE	

Understanding Why Macro Quoting is Necessary

Macro quoting functions tell the macro processor to interpret special characters and mnemonics as text rather than as part of the macro language. If you did not use a macro quoting function to mask the special characters, the macro processor or the rest of the SAS System might give the character a meaning you did not intend. Here are some examples of the kinds of ambiguities that can arise when text strings contain special characters and mnemonics:

- Is **`%sign`** a call to the macro SIGN or a phrase "percent sign"?
- Is OR the mnemonic Boolean operator or the abbreviation for Oregon?
- Is the quote in O'Malley an unbalanced single quotation mark or just part of the name?
- Is Boys&Girls a reference to the macro variable &GIRLS or a group of children?
- Is GE the mnemonic for "greater than or equal" or is it short for General Electric?
- Which statement does a semicolon end?
- Does a comma separate parameters, or is it part of the value of one of the parameters?

Macro quoting functions enable you to clearly indicate to the macro processor how it is to interpret special characters and mnemonics.

Here is an example, using the simplest macro quoting function, %STR. Suppose you want to assign a PROC PRINT statement and a RUN statement to the macro variable PRINT. Here is the erroneous statement:

```
%let print=proc print; run;;  /* ERROR */
```

This code is ambiguous. Are the semicolons that follow PRINT and RUN part of the value of the macro variable PRINT, or does one of them end the %LET statement? If you do not tell the macro processor what to do, it interprets the semicolon after PRINT as the end of the %LET statement; the value of the PRINT macro variable is

```
proc print
```

The rest of the characters (RUN;;) are simply the next part of the program.

To avoid the ambiguity and correctly assign the value of PRINT, you must mask the semicolons with the macro quoting function %STR, as follows:

```
%let print=%str(proc print; run;);
```

Understanding Macro Quoting Functions

The following macro quoting functions are most commonly used:

- %STR and %NRSTR
- %BQUOTE and %NRBQUOTE
- %SUPERQ

For the paired macro quoting functions, the function beginning with NR affects the same category of special characters that are masked by the plain macro quoting function as well as ampersands and percent signs. In effect, the NR functions prevent macro and macro variable resolution. To help you remember which does which, try associating the NR in the macro quoting function names with the words "not resolved" — that is, macros and macro variables are not resolved when you use these functions.

The macro quoting functions with B in their names are useful for macro quoting unmatched quotation marks and parentheses. As a help for remembering this, try associating B with "by itself".

The %SUPERQ macro quoting function is unlike the other macro quoting functions in that it does not have a mate and works differently than the other macro quoting functions. See "Using %SUPERQ" on page 82 for more information on the %SUPERQ macro quoting function.

The macro quoting functions can also be divided into three types, depending on when they take effect:

compilation functions	cause the macro processor to interpret special characters as text in a macro program statement in open code or while compiling (constructing) a macro. The %STR and %NRSTR functions are compilation functions.
execution functions	cause the macro processor to treat as text special characters that result from resolving a macro expression (such as a macro variable reference, a macro invocation, or the argument of an implicit %EVAL function). They are called execution functions because resolution occurs during macro execution or during execution of a macro program statement in open code. The macro processor resolves the expression as far as possible, issues any warning messages for macro variable references or macro invocations it cannot resolve, and quotes the result. The %BQUOTE and %NRBQUOTE functions are execution functions. An *implicit %EVAL function* is one that is caused by another macro statement, such as a %DO %TO statement. An *explicit %EVAL function* is one that is directly called by macro code.
function that prevents resolution	causes the macro processor to treat the value of a macro variable as a "picture" during macro execution. It treats the value as text, without beginning the process of resolution. The %SUPERQ function prevents resolution of a macro variable's value. The %SUPERQ function takes as its argument a macro variable name (or a macro expression that yields a macro variable name). The argument must not be a reference to the macro variable whose value you are masking; that is, do not include the & before the name.

Note: Two other execution macro quoting functions exist, %QUOTE and %NRQUOTE. They are useful for unique macro quoting needs and for compatibility with older macro applications. For more information on these two macro quoting functions, refer to Chapter 13, "Macro Language Dictionary." △

Passing Parameters That Contain Special Characters and Mnemonics

Using an execution macro quoting function in the macro definition is the simplest and best way to have the macro processor accept resolved values that may contain special characters. However, if you discover that you need to pass parameter values such as **or** when a macro has not been defined with an execution macro quoting function, you can do so by masking the value in the macro invocation. The logic of the process is as follows:

1. When you mask a special character with a macro quoting function, it remains masked as long as it is within the macro facility (unless you use the %UNQUOTE function, described in "nquoting Text"later in this chapter).
2. The macro processor constructs the complete macro invocation before beginning to execute the macro.
3. Therefore, you can mask the value in the invocation with the %STR function. Although the masking is not needed when the macro processor is constructing the invocation, the value is already masked by a macro quoting function when macro execution begins and therefore does not cause problems during macro execution.

For example, suppose a macro named ORDERX does not use the %BQUOTE function. You can pass the value **or** to the ORDERX macro with the following invocation:

```
%orderx(%str(or))
```

However, placing the macro quoting function in the macro definition makes the macro much easier for you to invoke.

Deciding When to Use a Macro Quoting Function and Which Function to Use

Use a macro quoting function anytime you want to assign to a macro variable a special character that could be interpreted as part of the macro language. Table 7.1 on page 74 describes the special characters to mask when used as part of a text string and which macro quoting functions are useful in each situation.

Table 7.1 Special Characters and When Macro Quoting is Needed

Special Character...	Must Be Masked...	Quoted by All Macro Quoting Functions?	Remarks
+-*/<>=^|¬ ~ LE LT EQ NE GE GT AND OR NOT	to prevent it from being treated as an operator in the argument of an explicit or implicit %EVAL function	yes	AND, OR, and NOT need to be masked because they are interpreted as mnemonic operators by an implicit %EVAL and by %SYSEVALF.
blank	to maintain, rather than ignore, a leading, trailing, or isolated blank	yes	

Special Character...	Must Be Masked...	Quoted by All Macro Quoting Functions?	Remarks
;	to prevent a macro program statement from ending prematurely	yes	
, (comma)	to prevent it from indicating a new function argument, parameter, or parameter value	yes	
' " ( )	if it may be unmatched	no	Arguments that may contain quotation marks and parentheses should be masked with a macro quoting function so that the macro facility interprets the single and double quotation marks and parentheses as text rather than macro language symbols or possibly unmatched quotation marks or parentheses for the SAS language. With %STR, %NRSTR, %QUOTE, and %NRQUOTE, unmatched quotation marks and parentheses must be marked with a % sign. You do not have to mark unmatched symbols in the arguments of %BQUOTE, %NRBQUOTE, and %SUPERQ.
%*name* &*name*	(depends on what the expression might resolve to)	no	%NRSTR, %NRBQUOTE, and %NRQUOTE mask these patterns. To use %SUPERQ with a macro variable, omit the ampersand from *name*.

The macro facility allows you as much flexibility as possible in designing your macros. You need to mask a special character with a macro quoting function only when the macro processor would otherwise interpret the special character as part of the

macro language rather than as text. For example, in this statement you must use a macro quoting function to mask the first two semicolons to make them part of the text:

```
%let p=%str(proc print; run;);
```

However, in the macro PR, shown here, you do not need to use a macro quoting function to mask the semicolons after PRINT and RUN:

```
%macro pr(start);
   %if &start=yes %then
      %do;
         %put proc print requested;
         proc print;
         run;
      %end;
%mend pr;
```

Because the macro processor does not expect a semicolon within the %DO group, the semicolons after PRINT and RUN are not ambiguous, and they are interpreted as text.

Although it is not possible to give a series of rules that cover every situation, the following sections describe how to use each macro quoting function. Table 7.4 on page 85, later in this chapter, provides a summary of the various characters that may need masking and of which macro quoting function is useful in each situation.

Note: You can also perform the inverse of a macro quoting function–that is, remove the tokenization provided by macro quoting functions. For an example of when the %UNQUOTE function is useful, see "Unquoting Text" on page 85. △

Using the %STR and %NRSTR Functions

If a special character or mnemonic affects the way the macro processor constructs macro program statements, you must mask the item during macro compilation (or during the compilation of a macro program statement in open code) by using either the %STR or %NRSTR macro quoting functions.

These macro quoting functions mask the following special characters and mnemonics:

blank	)	=	LT
;	(	|	GE
¬	+	AND	GT
^	—	OR	
~	*	NOT	
, (comma)	/	EQ	
'	<	NE	
"	>	LE	

In addition to these, %NRSTR masks & and %.

Note: If an unmatched single or double quotation mark or a left or right parenthesis is used with %STR or %NRSTR, these characters must be preceded by a percent sign (%). See "Using Unmatched Quotation Marks and Parentheses with %STR and %NRSTR" later in this chapter for more information. △

When you use %STR or %NRSTR, the macro processor does not receive these functions and their arguments when it executes a macro. It receives only the results of these functions because these functions work when a macro compiles. This means by the time the macro executes, the string is already masked by a macro quoting function. Therefore, %STR and %NRSTR are useful for masking strings that are constants, such as sections of SAS code. In particular, %NRSTR is a good choice for masking strings that contain % and & signs. However, these functions are not so useful for masking strings that contain references to macro variables because it is possible that the macro variable could resolve to a value not quotable by %STR or %NRSTR. For example, the string could contain an unmarked, unmatched left parenthesis.

Using Unmatched Quotation Marks and Parentheses with %STR and %NRSTR

If the argument to %STR or %NRSTR contains an unmatched single or double quotation mark or an unmatched left or right parenthesis, precede each of these characters with a % sign. Table 7.2 on page 77 shows some examples of this technique.

Table 7.2 Examples of Marking Unmatched Quotation Marks and Parenthese with %STR and %NRSTR

Notation	Description	Example	Quoted Value Stored
%'	unmatched single quotation mark	`%let myvar=%str(a%');`	`a'`
%"	unmatched double quotation mark	`%let myvar=%str(title %''first);`	`title ''first`
%(	unmatched left parenthesis	`%let myvar=%str (log%(12);`	`log(12`
%)	unmatched right parenthesis	`%let myvar=%str (345%));`	`345)`

Using % Signs with %STR

In general, if you want to mask a % sign with a macro quoting function at compilation, use %NRSTR. There is one case where you can use %STR to mask a % sign: when the % sign does not have any text following it that could be construed by the macro processor as a macro name. The % sign must be marked by another % sign. Here are some examples.

Table 7.3 Examples of Masking % Signs with %STR

Notation	Description	Example	Quoted Value Stored
'%'	% sign before a matched single quotation mark	`%let myvar=%str('%');`	'%'
%%%'	% sign before an unmatched single quotation mark	`%let myvar=%str(%%%');`	`%'`

Notation	Description	Example	Quoted Value Stored
""%%	% sign after a matched double quotation mark	`%let myvar=%str(""%%);`	`""%`
%%%%	two % signs in a row	`%let myvar=%str(%%%%);`	`%%`

Examples Using %STR

The %STR function in the following %LET statement prevents the semicolon after PROC PRINT from being interpreted as the ending semicolon for the %LET statement:

```
%let printit=%str(proc print; run;);
```

As a more complex example, the macro KEEPIT1 shows how the %STR function works in a macro definition:

```
%macro keepit1(size);
   %if &size=big %then %str(keep city _numeric_;);
   %else %str(keep city;);
%mend keepit1;
```

Call the macro as follows:

```
%keepit1(big)
```

This produces the following statement:

```
keep city _numeric_;
```

When you use the %STR function in the %IF-%THEN statement, the macro processor interprets the first semicolon after the word %THEN as text. The second semicolon ends the %THEN statement, and the %ELSE statement immediately follows the %THEN statement. Thus, the macro processor compiles the statements as you intended. However, if you omit the %STR function, the macro processor interprets the first semicolon after the word %THEN as the end of the %THEN clause and the next semicolon as constant text. Because constant text cannot appear between a %THEN and a %ELSE clause, the macro processor does not compile the macro. Instead, it issues an error message.

In the %ELSE statement, the %STR function causes the macro processor to treat the first semicolon in the statement as text and the second one as the end of the %ELSE clause. Therefore, the semicolon that ends the KEEP statement is part of the conditional execution. If you omit the %STR function, the first semicolon ends the %ELSE clause and the second semicolon is outside the conditional execution. It is generated as text each time the macro executes. (In this example, the placement of the semicolon does not affect the SAS code.) Again, using %STR causes the macro KEEPIT1 to compile as you intended.

Here is an example that uses %STR to mask a string that contains an unmatched single quotation mark. Note the use of the % sign before the quotation mark:

```
%let innocent=%str(I didn%'t do it!);
```

Examples Using %NRSTR

Suppose you want the name (not the value) of a macro variable to be printed by the %PUT statement. To do so, you must use the %NRSTR function to mask the & and prevent the resolution of the macro variable, as in the following example:

```
%macro example;
   %local myvar;
   %let myvar=abc;
   %put %nrstr(The string &myvar appears in log output,);
   %put instead of the variable value.;
%mend example;

%example
```

This code writes the following text to the SAS log:

```
The string &myvar appears in log output,
instead of the variable value.
```

If you did not use the %NRSTR function or if you used %STR, the following undesired output would appear in the SAS log:

```
The string abc appears in log output,
instead of the variable value.
```

The %NRSTR function prevents the & from triggering macro variable resolution.

The %NRSTR function is also useful when the macro definition contains patterns that the macro processor would ordinarily recognize as macro variable references, as in the following program:

```
%macro credits(d=%nrstr(Mary&Stacy&Joan Ltd.));
   footnote "Designed by &d";
%mend credits;
```

Using %NRSTR causes the macro processor to treat &STACY and &JOAN simply as part of the text in the value of D; the macro processor does not issue warning messages for unresolvable macro variable references. Suppose you invoke the macro CREDITS with the default value of D, as follows:

```
%credits()
```

Submitting this program generates the following FOOTNOTE statement:

```
footnote "Designed by Mary&Stacy&Joan Ltd.";
```

If you omit the %NRSTR function, the macro processor attempts to resolve the references &STACY and &JOAN as part of the resolution of &D in the FOOTNOTE statement. The macro processor issues these warning messages (assuming the SERROR system option, described in Chapter 13, is active) because no such macro variables exist:

```
WARNING: Apparent symbolic reference STACY not resolved.
WARNING: Apparent symbolic reference JOAN not resolved.
```

Here is a final example of using %NRSTR. Suppose you wanted to have a text string include the name of a macro function: **`This is the result of %NRSTR`**. Here is the program:

```
%put This is the result of %nrstr(%nrstr);
```

You must use %NRSTR to mask the % sign at compilation, so the macro processor does not try to invoke %NRSTR a second time. If you did not use %NRSTR to mask the string **`%nrstr`**, the macro processor would complain about a missing open parenthesis for the function.

Using the %BQUOTE and %NRBQUOTE Functions

%BQUOTE and %NRBQUOTE mask values during execution of a macro or a macro language statement in open code. These functions instruct the macro processor to resolve a macro expression as far as possible and mask the result, issuing any warning messages for macro variable references or macro invocations it cannot resolve. These functions mask all the characters that %STR and %NRSTR mask with the addition of unmarked percent signs; unmatched, unmarked single and double quotation marks; and unmatched, unmarked opening and closing parentheses. That means that you do not have to precede an unmatched quotation mark with a % sign, as you must when using %STR and %NRSTR.

The %BQUOTE function treats all parentheses and quotation marks produced by resolving macro variable references or macro calls as special characters to be masked by a macro quoting function. (It does not mask parentheses or quotation marks that are not produced by resolution.) Therefore, it does not matter whether quotation marks and parentheses in the resolved value are matched; each one is masked individually.

The %NRBQUOTE function is useful when you want a value to be resolved when first encountered, if possible, but you do not want any ampersands or percent signs in the result to be interpreted as operators by an explicit or implicit %EVAL function.

If the argument of the %NRBQUOTE function contains an unresolvable macro variable reference or macro invocation, the macro processor issues a warning message before it masks the ampersand or percent sign (assuming the SERROR or MERROR system option, described in Chapter 13, is in effect). To suppress the message for unresolved macro variables, use the %SUPERQ function (discussed later in this chapter) instead.

Because the %BQUOTE and %NRBQUOTE functions operate during execution and are more flexible than %STR and %NRSTR, %BQUOTE and %NRBQOUTE are good choices for masking strings which contain macro variable references.

Examples Using %BQUOTE

In the following statement, the %IF-%THEN statement uses %BQUOTE to prevent an error if the macro variable STATE resolves to OR (for Oregon), which the macro processor would interpret as the logical operator OR otherwise:

```
%if %bquote(&state)=OR %then %put Oregon Dept. of Revenue;
```

Note: This example works if you use %STR–but it is not robust or good programming practice. Because you cannot guarantee what &STATE is going to resolve to, you need to use %BQUOTE to mask the resolution of the macro variable at execution time, not the name of the variable itself at compile time. △

In the following example, a DATA step creates a character value containing a single quotation mark and assigns that value to a macro variable. The macro READIT then uses the %BQUOTE function to allow a %IF condition to accept the unmatched single quotation mark:

```
data test;
   store="Susan's Office Supplies";
   call symput('s',store);
run;

%macro readit;
```

```
   %if %bquote(&s) ne %then %put *** valid ***;
   %else %put *** null value ***;
%mend readit;

%readit
```

When you assign the value **Susan's Office Supplies** to STORE in the DATA step, enclosing the character string in double quotation marks allows you to use an unmatched single quotation mark in the string. The SAS System stores the value of STORE as

```
Susan's Office Supplies
```

The CALL SYMPUT routine assigns that value (containing an unmatched single quotation mark) as the value of the macro variable S. If you do not use the %BQUOTE function when you reference S in the macro READIT, the macro processor issues an error message for an invalid operand in the %IF condition.

When you submit the code, the following is written to the SAS log:

```
*** valid ***
```

Referring to Already Quoted Variables

Items that have been masked by a macro quoting function, such as the value of &WHOSE in the following program, remain masked as long as the item is being used by the macro processor. When you use the value of WHOSE later in a macro program statement, you do not need to mask the reference again.

```
/* Use %STR to mask the constant, and use a  % sign to mark */
/* the unmatched single quotation mark. */
%let whose=%str(John%'s);

/* You don't need to mask the macro reference, since it was */
/* masked in the %LET statement, and remains masked. */
%put *** This coat is &whose ***;
```

Deciding How Much Text to Mask with a Macro Quoting Function

In each of the following statements, the macro processor treats the masked semicolons as text:

```
%let p=%str(proc print; run;);
%let p=proc %str(print;) %str(run;);
%let p=proc print%str(;) run%str(;);
```

The value of P is the same in each case:

```
proc print; run;
```

The results of the three %LET statements are the same because when you mask text with a macro quoting function, the macro processor quotes only the items that the function recognizes. Other text enclosed in the function remains unchanged. Therefore, the third %LET statement is the minimalist approach to macro quoting. However, masking "too much" text with a macro quoting function is harmless and results in code that is much easier to read (such as the first %LET statement).

Using %SUPERQ

The %SUPERQ function locates the macro variable named in its argument and quotes the value of that macro variable without permitting any resolution to occur. It masks all items that may require macro quoting at macro execution. Because %SUPERQ does not attempt any resolution of its argument, the macro processor does not issue any warning messages that a macro variable reference or a macro invocation has not been resolved. Therefore, even when the %NRBQUOTE function allows the program to work correctly, you can use the %SUPERQ function to eliminate unwanted warning messages from the SAS log. %SUPERQ takes as its argument a macro variable name without an ampersand or a text expression that yields a macro variable name.

%SUPERQ retrieves the value of a macro variable from the macro symbol table and quotes it immediately, preventing the macro processor from making any attempt to resolve anything that may occur in the resolved value. For example, if the macro variable CORPNAME resolves to **Smith&Jones**, using %SUPERQ prevents the macro processor from attempting to further resolve **&Jones**. This %LET statement successfully assigns the value **Smith&Jones** to TESTVAR:

```
%let testvar=%superq(corpname);
```

Examples Using %SUPERQ

This example shows how the %SUPERQ function affects two macro invocations, one for a macro that has been defined and one for an undefined macro:

```
%macro a;
   %put *** This is a. ***;
%mend a;

%macro test;
   %put *** Enter two values: ***;
   %input;
   %put *** %superq(sysbuffr) ***;    /* Note absence of ampersand */
%mend test;
```

Suppose you invoke the macro TEST and respond to the prompt as shown:

```
%test
*** Enter two values: ***
%a %x
```

The second %PUT statement simply writes the following line:

```
*** %a %x ***
```

It does not invoke the macro A, and it does not issue a warning message that %X was not resolved. See Chapter 13 for a description of SYSBUFFR.

The following two examples compare the %SUPERQ function with other macro quoting functions.

Using the %SUPERQ Function to Prevent Warning Messages

The discussions of the %NRBQUOTE function showed that this function causes the macro processor to attempt to resolve the patterns *&name* and *%name* the first time it encounters them; if the macro processor cannot resolve them, it quotes the ampersand or percent sign so that later uses of the value do not cause the macro processor to

recognize them. However, if the MERROR or SERROR option is in effect, the macro processor issues a warning message that the reference or invocation was not resolved.

The macro FIRMS3, shown here, shows how the %SUPERQ function can prevent unwanted warning messages:

```
%macro firms3;
   %global code;
   %put Enter the name of the company;
   %input;
   %let name=%superq(sysbuffr);
   %if &name ne %then %let code=valid;
   %else %let code=invalid;
   %put *** &name is &code ***;
%mend firms3;
```

Suppose you invoke the macro FIRMS3 twice and respond with the companies shown here:

```
A&A Autos
Santos&D'Amato
```

After the macro executes, the following is written to the SAS log:

```
*** A&A Autos is valid ***
*** Santos&D'Amato is valid ***
```

Using the %SUPERQ Function to Enter Macro Keywords

Suppose you create an online training system in which users can enter problems and questions that another macro prints for you later. The user's response to a %INPUT statement is assigned to a local macro variable and then to a global macro variable. Because the user is asking questions about macros, he or she may enter all sorts of macro variable references and macro calls as examples of problems, as well as unmatched, unmarked quotation marks and parentheses. If you mask the response with %BQUOTE, you have to use a few %PUT statements to warn the user about responses that cause problems. If you use the %SUPERQ function, you need fewer instructions. The macros ASK1 and ASK2 show how the macro code becomes simpler as you change macro quoting functions.

The macro ASK1, below, shows how the macro looks when you use the %BQUOTE function:

```
%macro ask1;
   %global myprob;
   %local temp;
   %put Describe the problem.;
   %put Do not use macro language keywords, macro calls,;
   %put or macro variable references.;
   %put Enter /// when you are finished.;
   %do %until(%bquote(&sysbuffr) eq %str(///));
      %input;
      %let temp=&temp %bquote(&sysbuffr);
   %end;
   %let myprob=&temp;
%mend ask1;
```

The macro ASK1 does not include a warning about unmatched quotation marks and parentheses. You can invoke the macro ASK1 and enter a problem as shown:

```
%ask1
Describe the problem.
Do not use macro language keywords, macro calls,
or macro variable references.
Enter /// when you are finished.
Why didn't my macro run when I called it? (It had three
parameters, but I wasn't using any of them.) It ran
after I submitted the next statement.
///
```

Notice that both the first and second lines of the response contain an unmatched, unmarked quotation mark and parenthesis. %BQUOTE can handle these characters.

The macro ASK2, shown here, modifies the macro ASK1 by using the %SUPERQ function. Now the %INPUT statement accepts macro language keywords and does not attempt to resolve macro calls and macro variable references:

```
%macro ask2;
   %global myprob;
   %local temp;
   %put Describe the problem.;
   %put Enter /// when you are finished.;
   %do %until(%superq(sysbuffr) eq %str(///)); /* No ampersand */
      %input;
      %let temp=&temp %superq(sysbuffr);   /* No ampersand */
   %end;
   %let myprob=&temp;
%mend ask2;
```

You can invoke the macro ASK2 and enter a response as shown:

```
%ask2
Describe the problem.
Enter /// when you are finished.
My macro ADDRESS starts with %MACRO ADDRESS(COMPANY,
CITY);. I called it with %ADDRESS(SMITH-JONES, INC., BOSTON),
but it said I had too many parameters. What happened?
///
```

The response contains a macro language keyword, a macro invocation, and unmatched parentheses.

Summary of Macro Quoting Functions and the Characters They Mask

Different macro quoting functions mask different special characters and mnemonics so the macro facility interprets them as text instead of as macro language symbols.

Table 7.4 on page 85 divides the symbols into categories and shows which macro quoting functions mask which symbols.

By Item

Table 7.4 Summary of Special Characters and Macro Quoting Functions

Group	Items	Macro Quoting Functions
A	+ — */<>=¬^ | ~;, blank AND OR NOT EQ NE LE LT GE GT	all
B	&%	%NRSTR, %NRBQUOTE, %SUPERQ, NRQUOTE
C	unmatched' "()	%BQUOTE, %NRBQUOTE, %SUPERQ, %STR*, %NRSTR*, %QUOTE*, %NRQUOTE*

By Function

Function	Affects Groups	Works at
%STR	A, C*	macro compilation
%NRSTR	A, B, C*	macro compilation
%BQUOTE	A, C	macro execution
%NRBQUOTE	A, B, C	macro execution
%SUPERQ	A, B, C	macro execution (prevents resolution)
%QUOTE	A, C*	macro execution. Requires unmatched quotation marks and parentheses to be marked with a percent sign (%).
%NRQUOTE	A, B, C*	macro execution. Requires unmatched quotation marks and parentheses to be marked with a percent sign (%).

*Unmatched quotation marks and parentheses must be marked with a percent sign (%) when used with %STR, %NRSTR, %QUOTE, and %NRQUOTE.

Unquoting Text

To *unquote* a value means to restore the significance of symbols in an item that was previously masked by a macro quoting function.

Usually, after an item has been masked by a macro quoting function, it retains its special status until one of the following occurs:

- You enclose the item with the %UNQUOTE function (described in Chapter 13).
- The item leaves the word scanner and is passed to the DATA step compiler, SAS procedures, or other parts of the SAS System, when the item is part of generated SAS statements.
- The item is returned as an unquoted result by the %SCAN, %SUBSTR, or %UPCASE function. (To retain a value's masked status during one of these operations, use the %QSCAN, %QSUBSTR, or %QUPCASE function. See "Other Functions That Perform Macro Quoting" on page 88 for more details.)

As a rule, you do not need to unquote an item because it is automatically unquoted when the item is passed from the word scanner to the rest of the SAS System. Under two circumstances, however, you may need to use the %UNQUOTE function to restore the original significance to a masked item:

- when you want to use a value with its restored meaning later in the same macro in which its value was previously masked by a macro quoting function.
- when, as in a few cases, masking text with a macro quoting function changes the way the word scanner tokenizes it, producing SAS statements that look correct but that the SAS compiler does not recognize.

Example of Unquoting

The following example illustrates using a value twice: once in macro quoted form and once in unquoted form. Suppose the macro ANALYZE is part of a system that allows you to compare the output of two statistical models interactively. First, you enter an operator to specify the relationship you want to test (one result greater than another, equal to another, and so forth). The macro ANALYZE tests the macro quoted value of the operator to verify that you have entered it correctly, uses the unquoted value to compare the values indicated, and writes a message. Match the numbers in the comments to the paragraphs below.

```
%macro analyze(stat);
   data _null_;
      set out1;
      call symput('v1',&stat);
   run;

   data _null_;
      set out2;
      call symput('v2',&stat);
   run;

   %put Preliminary test. Enter the operator.;
   %input;
   %let op=%bquote(&sysbuffr);     ❶
   %if &op=%str(=<) %then %let op=%str(<=);  ❷    ❸
   %else %if &op=%str(=>) %then %let op=%str(>=);
   %if &v1 %unquote(&op) &v2 %then   ❹
      %put You may proceed with the analysis.;
   %else
      %do;
         %put &stat from out1 is not &op &stat from out2.;
         %put Please check your previous models.;
      %end;
%mend analyze;
```

You mask the value of SYSBUFFR with the %BQUOTE function, which masks resolved items including unmatched, unmarked quotation marks and parentheses (but excluding the ampersand and percent sign).

The %IF condition compares the value of the macro variable OP to a string to see whether the value of OP contains the correct symbols for the operator. If the value contains symbols in the wrong order, the %THEN statement corrects the symbols. Because a value masked by a macro quoting function remains masked, you do not need to mask the reference &OP in the left side of the %IF condition.

Because you can see the characters in the right side of the %IF condition and in the %LET statement when you define the macro, you can use the %STR function to mask them. Masking them once at compilation is more efficient than masking them at each execution of ANALYZE.

To use the value of the macro variable OP as the operator in the %IF condition, you must restore the meaning of the operator with the %UNQUOTE function.

What to Do When Automatic Unquoting Does Not Work

When the macro processor generates text from an item masked by a macro quoting function, you can usually allow the SAS System to unquote the macro quoted items automatically. For example, suppose you define a macro variable PRINTIT as follows:

```
%let printit=%str(proc print; run;);
```

Then you use that macro variable in your program like this:

```
%put *** This code prints the data set: &printit ***;
```

When the macro processor generates the text from the macro variable, the items masked by macro quoting functions are automatically unquoted, and the previously masked semicolons work normally when they are passed to the rest of the SAS System.

In rare cases, masking text with a macro quoting function changes the way the word scanner tokenizes the text. (The word scanner and tokenization are discussed in Chapter 2, "SAS Programs and Macro Processing" and Chapter 4, "Macro Processing.") For example, a single or double quotation mark produced by resolution within the %BQUOTE function becomes a separate token; the word scanner does not use it as the boundary of a literal token in the input stack. If generated text that was once masked by the %BQUOTE function looks correct but the SAS System does not accept it, you may need to use the %UNQUOTE function to restore normal tokenization.

Understanding How Macro Quoting Works "Behind the Scenes"

In simple terms, when the macro processor masks a text string, it masks special characters and mnemonics within the coding scheme, and prefixes and suffixes the string with a hexadecimal character, called a *delta character*. The prefix character marks the beginning of the string and also indicates what type of macro quoting is to be applied to the string. The suffix character marks the end of the string. The prefix and suffix characters preserve any leading and trailing blanks contained by the string. The hexadecimal characters used to mask special characters and mnemonics and those used for the prefix and suffix characters may vary and are not portable.

There are more hexadecimal combinations possible in each byte than are needed to represent the symbols on a keyboard. Therefore, when a macro quoting function recognizes an item to be masked, the macro processor uses a previously unused hexadecimal combination for the prefix and suffix characters.

Macro functions, such as %EVAL and %SUBSTR, ignore the prefix and suffix characters. Therefore, the prefix and suffix characters do not affect comparisons.

When the macro processor is finished with a macro quoted text string, it removes the macro quoting-coded substitute characters and replaces them with the original characters. The unmasked characters are passed on to the rest of the system. Sometimes you may see a message about this unmasking, as in the following example:

```
/* Turn on SYMBOLGEN so you can see the messages about unquoting. */
options symbolgen;
```

```
/* Assign a value to EXAMPLE that contains several special */
/* characters and a mnemonic. */
%let example = %nrbquote( 1 + 1 = 3 Today's Test and More );

%put *&example*;
```

When this program is submitted, the following appears in the SAS log:

```
SYMBOLGEN:   Macro variable EXAMPLE resolves to  1 + 1 = 3 Today's
             Test and More
SYMBOLGEN:   Some characters in the above value which were subject
             to macro quoting have been unquoted for printing.
* 1 + 1 = 3 Today's Test and More *
```

As you can see, the leading and trailing blanks and special characters were retained in the variable's value. While the macro processor was working with the string, the string actually contained coded characters that were substituted for the "real" characters. The substitute characters included coded characters to represent the start and end of the string. This preserved the leading and trailing blanks. Characters were also substituted for the special characters **+**, **=**, and **'**, and the mnemonic **AND**. When the macro finished processing and the characters were passed to the rest of the SAS System, the coding was removed and the real characters were replaced.

"Unquoting Text" on page 85 provides more information on what happens when a masked string is unquoted. Chapter 13 describes the SYMBOLGEN system option.

Other Functions That Perform Macro Quoting

Some macro functions are available in pairs, where one function starts with the letter Q:

- %SCAN and %QSCAN
- %SUBSTR and %QSUBSTR
- %UPCASE and %QUPCASE
- %SYSFUNC and %QSYSFUNC.

The Q*xxx* functions are necessary because by default, macro functions return an unquoted result, even if the argument was masked by a macro quoting function. The %QSCAN, %QSUBSTR, %QUPCASE, and %QSYSFUNC functions mask the returned value. The items masked are the same as those masked by the %NRBQUOTE function.

Example Using the %QSCAN Function

The following macro uses the %QSCAN function to assign items in the value of SYSBUFFR (described in Chapter 13) as the values of separate macro variables. The numbers in the comments correspond to the explanations in the list that follows the macro code.

```
%macro splitit;
   %put What character separates the values?;  ❶
   %input;
   %let s=%bquote(&sysbuffr);    ❷
   %put Enter three values.;
   %input;
```

```
   %local i;
   %do i=1 %to 3;  ❸
      %global x&i;
      %let x&i=%qscan(%superq(sysbuffr),&i,&s);  ❹
   %end;
%mend splitit;

%splitit
What character separates the values?
#
Enter three values.
Fischer Books#Smith&Sons#Sarah's Sweet Shoppe  ❺
```

1 This question asks you to input a delimiter for the %QSCAN function that will not appear in the values you are going to enter.

2 Masking the value of SYSBUFFR with the %BQUOTE function allows you to choose a quotation mark or parenthesis as a delimiter if necessary.

3 The iterative %DO loop creates a global macro variable for each segment of SYSBUFFR and assigns it the value of that segment.

4 The %SUPERQ function masks the value of SYSBUFFR in the first argument of the %QSCAN function. It prevents any resolution of the value of SYSBUFFR.

5 The %QSCAN function returns macro quoted segments of the value of SYSBUFFR; thus, the unmatched quotation mark in **`Sarah's Sweet Shoppe`** and the &*name* pattern in **`Smith&Sons`** do not cause problems.

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