Introduction

This chapter describes a typical pattern that the SAS System follows to process a program. These concepts are helpful in understanding how the macro processor works with other parts of the SAS System. However, they are not required for most macro programming. They are provided so that you can understand what is going on behind the scenes.

Note: The concepts in this chapter present a logical representation, not a detailed physical representation, of how SAS software works.

When you submit a program, it goes to an area of memory called the input stack. This is true for all program and command sources: the Display Manager, the SCL SUBMIT block, the SCL COMPILE command, or from batch or noninteractive sessions. The input stack shown in Figure 2.1 on page 10 contains a simple SAS program that displays sales data. The first line in the program is the top of the input stack.
How SAS Processes Statements without Macro Activity

The process that SAS uses to extract words and symbols from the input stack is called tokenization. Tokenization is performed by a component of SAS called the word scanner, as shown in Figure 2.2 on page 11. The word scanner starts at the first character in the input stack and examines each character in turn. In doing so, the word scanner assembles the characters into tokens. There are four general types of tokens:

- **Literal**: a string of characters enclosed in quotation marks.
Number
digits, date values, time values, and hexadecimal numbers.

Name
a string of characters beginning with an underscore or letter.

Special
any character or group of characters that have special meaning to the SAS System, for example SAS operators. Examples of special characters include:

* / + - ** ; $ ( ) . & % =

For more information on tokens, see Appendix 2, "SAS Tokens."

Figure 2.2 The Sample Program before Tokenization

<table>
<thead>
<tr>
<th>Word Scanner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>data sales (drop=lastyr);</td>
</tr>
<tr>
<td>infile inl;</td>
</tr>
<tr>
<td>input ml-ml2 lastyr;</td>
</tr>
<tr>
<td>total=ml2+lastyr;</td>
</tr>
<tr>
<td>run;</td>
</tr>
</tbody>
</table>

The first SAS statement in the input stack (Figure 2.2 on page 11) contains eight tokens (four names and four special characters).

data sales(drop=lastyr);

When the word scanner finds a blank or the beginning of a new token, it removes a token from the input stack and transfers it to the bottom of the queue.

In this example, when the word scanner pulls the first token from the input stack, it recognizes the token as the beginning of a DATA step. The word scanner triggers the DATA step compiler, which begins to request more tokens. The compiler pulls tokens from the top of the queue, as shown in Figure 2.3 on page 12.
How SAS Processes Statements with Macro Activity

In a program with macro activity, the macro processor can generate text that is placed on the input stack to be tokenized by the word scanner. The example in this section shows you how the macro processor creates and resolves a macro variable. To illustrate how the compiler and the macro processor work together, Figure 2.5 on page 13 contains the macro processor and the macro variable symbol table. SAS creates the symbol table at the beginning of a SAS session to hold the values of automatic and global macro variables. SAS creates automatic macro variables at the beginning of a SAS session. For the sake of illustration, the symbol table is shown with only one automatic macro variable, SYSDAY.
Figure 2.5  The Macro Processor and Symbol Table

Whenever the word scanner encounters a macro trigger, it sends information to the macro processor. A macro trigger is either an ampersand (&) or percent sign (%) followed by a nonblank character. As it did in the previous example, the word scanner begins to process this program by examining the first characters in the input stack. In this case, the word scanner finds a percent sign (%) followed by a nonblank character. The word scanner recognizes this combination of characters as a potential macro language element, and triggers the macro processor to examine the tokens % and then LET, as shown in Figure 2.6 on page 13.

Figure 2.6  The Macro Processor Examines LET

When the macro processor recognizes a macro language element, it begins to work with the word scanner. In this case, the macro processor removes the %LET statement, and writes an entry in the symbol table, as shown in Figure 2.7 on page 14.
From the time the word scanner triggers the macro processor until that macro processor action is complete, the macro processor controls all activity. While the macro processor is active, no activity occurs in the word scanner or the DATA step compiler.

When the macro processor is finished, the word scanner reads the next token (the DATA keyword in this example) and sends it to the compiler. The word scanner triggers the compiler, which begins to pull tokens from the top of the queue, as shown in Figure 2.8 on page 14.

As it processes each token, SAS removes the protection that the macro quoting functions provide to mask special characters and mnemonic operators. For more information, see Chapter 7, “Macro Quoting.”
If the word scanner finds an ampersand followed by a nonblank character in a token, it triggers the macro processor to examine the next token, as shown in Figure 2.9 on page 15.

**Figure 2.9** The Macro Processor Examines &FILE

The macro processor examines the token and recognizes a macro variable that exists in the symbol table. The macro processor removes the macro variable name from the input stack and replaces it with the text from the symbol table, as shown in Figure 2.10 on page 15.

**Figure 2.10** The Macro Processor Generates Text to the Input Stack

The compiler continues to request tokens, and the word scanner continues to supply them, until the entire input stack has been read (Figure 2.11 on page 16).
If the end of the input stack is a DATA step boundary, as it is in this example, the compiler compiles and executes the step. SAS then frees the DATA step task. Any macro variables that were created during the program remain in the symbol table. If the end of the input stack is not a step boundary, the processed statements remain in the compiler. Processing resumes when more statements are submitted to the input stack.