Part 1

Macro Basics

Chapter 1 Introduction

This chapter introduces you to the fundamentals of the SAS macro language, and it includes an overview and some of the terminology of the language. Because the behavior of macros is different from that of code that is written for Base SAS, sections are also included on macro execution and how SAS sees and uses macros.

SEE ALSO
Stroupe (2003) and First (2003b) both do a nice job of providing an introduction to several of the basic concepts of the macro language.

1.1 Macro Facility Overview

The *SAS Macro Facility* is a tool within Base SAS software that contains the essential elements that enable you to use macros. The macro facility contains a *macro processor* that translates macro code into statements that can be used by SAS, and the macro language. The *macro language* provides the means to communicate with the macro processor.

The macro language consists of its own set of commands, options, syntax, and compiler. While many macro statements have similarities to the statements in the DATA step, you must understand the differences in behavior in order to effectively write and use macros.

The macro language provides tools that allow you to

- pass information between SAS steps
- dynamically create code after the user submits the program for execution
- conditionally execute DATA or PROC steps
- create generalizable and flexible code.

The tools made available through the macro facility include macro (or symbolic) variables, macro statements, and macro functions. These tools are included as part of the SAS code, or program, where they are detected when the code is sent to the SAS Supervisor for execution.

First and foremost, you should always keep in mind that the macro facility is a source code generator. Whether you are substituting the name of a data set or you are having the macro language write a complex DATA step, the macro facility will be writing code. It works with text as input and writes source code as output.

1.2 Terminology

The statement and syntax structure that is used by the macro facility is known as the *macro language* and like any language it has its own terminology. The SAS user who understands the programming language used in Base SAS, however, will discover quickly that much of the syntax and content of the macro language is familiar.

The following terms will be used throughout this book.

text

a collection of characters and symbols that can contain variable names, data set names, SAS statement fragments, complete SAS statements, or even complete DATA and PROC steps. Text forms the primary building blocks used by the macro language.

macro variable

the names of macro variables are almost always preceded by an ampersand (&) when used in SAS code. Macro variables are generally used to store text.

macro program statement

these statements control what actions take place during the macro execution. Like Base SAS language statements, they start with a keyword that in the macro language is always preceded by a percent sign (%), and are often syntactically similar to statements used in the DATA step.

macro facility

the software responsible for interpreting and executing macro language statements and elements.

macro references

when encountered by the SAS parser, these references to the macro language invoke the macro facility. The symbols & and % are used to designate these elements.

macro

also known as a macro program, a macro is a stored collection of macro language statements and text.

macro expression

one or more macro variable names, text, and/or macro functions combined together by one or more operators and/or parentheses. Macro expressions are very analogous to the expressions used in Base SAS programming.

macro function

predefined routines for processing text in macros and macro variables. Many macro functions are similar to functions used in the DATA step

operators

symbols that are used for comparisons, logical operation, or arithmetic calculations. The operators are the same ones used in base language comparisons.

automatic macro variable

special-purpose macro variables. These are automatically defined and provided by SAS. These variable names should be considered as reserved.

open code

SAS program statements that exist outside of any macro definition. Not all macro statements can be used in open code.

resolving macro references

during the resolution process, elements of the macro language (or references) are replaced with text.

When SAS statements are submitted for processing, they are broken up into their component parts so that SAS can understand them. This is done by the *word scanner*. The basic component parts are known as *tokens*. There are several types of these tokens, but two have special meaning to the macro language. These two tokens are the percent sign (%) and the ampersand (&). These tokens are macro processor *triggers*. When the word scanner detects one of these macro triggers (followed by a letter or underscore), the macro processor is invoked. The statement is then turned over to the macro facility for processing.

MORE INFORMATION
You can find additional terminology in the glossary.

SEE ALSO
Burlew (1998, Chapter 2) includes more information on macro language terminology.

1.3 Macro Execution Phases

When you run a SAS program, it is executed in a series of DATA and PROC steps, one step at a time. GLOBAL statements (for example, TITLE, FOOTNOTE, %LET), which can exist outside of these steps, are executed immediately when they are encountered. For each step, SAS first checks to see if macro references exist. *Macro references* may be macro variables, macro statements, macro definitions, or macro calls. If the program does not contain any macro references, then processing continues with the DATA or PROC step processor. If the program does contain macro references, then the macro processor intercepts and resolves them prior to execution. The resolved macro references then become part of the SAS code that is passed to the DATA or PROC step processor.

When code is passed to the SAS supervisor, the following takes place for each step:

- Global statements are executed.
- Macro definitions are compiled and stored until they are called.
- A check is made to see if there are any macro statements, macro variables, or macro calls. If there are, then
  - macro variables are resolved
  - called macros are executed (resolved)
  - macro statements are executed.
- The DATA or PROC step that contains resolved macro references (if there were any) is compiled and executed.

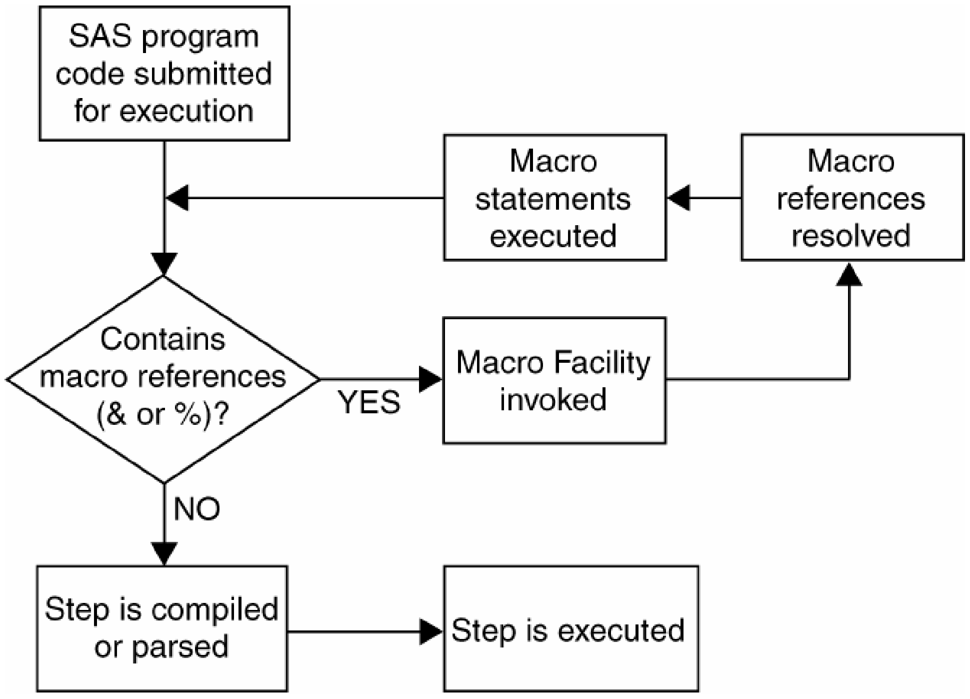
The following diagram is very much a simplification of the process described above. In fact, the SAS parser handles some of these steps in what is essentially a simultaneous manner. For ease of understanding the basic process, however, a simplification is called for, and an understanding of this process is absolutely essential if the programmer is to make full use of the macro language.

The most important information to glean from the diagram is the timing of the execution and resolution of macro language elements relative to Base SAS language elements. Not understanding this relationship causes new users to the macro language to ask questions like

- Can macro %IF statements be used interchangeably with DATA step IF statements?
- In a DATA step why can't I assign the value of a DATA step variable to a macro variable by using the %LET statement?
- Why can't I use a DATA step IF to conditionally execute a %LET?
- Why don't data set variables have values when using them in %IF statements?

Each of these questions can be answered by looking at this diagram. It shows the interaction between the macro and Base SAS language as well as the order that the different types of statements are executed.

The diagram is written as if it applies to each individual step. In fact, the same process applies at the statement and word level as well.


SAS program code submitted for execution
Macro statements executed
Macro references resolved
Contains macro references (& or %)?
YES
Macro Facility invoked
NO
Step is compiled or parsed
Step is executed

It is important for you to keep in mind that if there are Macro references in your code (% or &), these will be resolved **before** the step is even compiled.

Macro %IFs and DATA step IFs are not interchangeable because the %IF can **never** compare values of variables on the Program Data Vector (PDV). Indeed, the PDV does not yet exist when the %IF is executed. The %IF statement is described in Section 5.2.

For the same reason, you cannot conditionally assign a value to a macro variable using an IF statement and the %LET statement, because the %LET is a macro statement and is therefore executed long before the IF statement is even compiled. The %LET statement is first described in Section 2.2.

MORE INFORMATION

Additional comparisons between the macro language and the DATA step are made in Section 13.3.4.

SEE ALSO

SAS Macro Language: Reference, First Edition (pp. 14–19 and 33–41) and *SAS Macro Language: Reference, Version 8* (pp. 10–16 and 30–36) contain a detailed discussion of how SAS processes statements with macro activity. A very readable and detailed explanation of the internal processes of the macro facility from the SAS developer's point of view is offered by O'Connor (1998).

Jaffee (1999) restates this process in simple terms. Burlew (1998, Chapters 2 and 5) spends all of Chapter 2 and some of Chapter 5 discussing several variations on this series of events. A lot of detail is also included in Chapter 3 of *SAS Macro Language: Reference*, *First Edition*. First (2001) shows an example of the process as well as the timing of events.

1.4 Referencing Environments or Scopes

Unlike the values of data set variables, the values of macro variables are stored in memory in *symbol* tables. Each macro variable's definition in the symbol table is also associated with a *referencing environment* or *scope*, which is determined by where and how the macro variable is defined. There are two types of environments for macro variables: global and local. The terms "referencing environment" and "scope" are interchangeable, however "scope" is currently the preferred term.

A *global* macro variable has a single value available to all macros within the program. Macro variables that are defined outside of any macro will be global.

Local macro variables have values that are available only within the macro in which they are defined. Since macros can call other macros, there may be multiple levels of nested local tables.

Because each macro creates its own local scope, macro variable values that are defined in one macro may be undefined within another. Indeed, macro variable names need not be unique even among nested macros. This means that the specific value associated with a given macro variable may depend on how the macro variable is used in the program.

When macro calls are nested, their associated local symbol tables are also nested. This means that macro variables known to one macro may also be known within the macros that it calls.

In the following schematic, the macro variable DSN is defined globally and is, therefore, also known inside of the shaded macro. The macro variable COLOR, however, is only defined inside of the shaded macro and is not known outside of the macro.

```
Outside of all macros
Global values

DSN ---> clinics

Inside a macro
Local values

DSN ----> clinics
COLOR ---> blue

Outside of all macros

DSN ---> clinics
COLOR ---> undefined
```

MORE INFORMATION
You can control the referencing environment for a macro variable through the use of the %GLOBAL and %LOCAL statements, which are described in Section 5.4.2.

SEE ALSO

Extensive examples can be found in *SAS Guide to Macro Processing*, *Version 6*, *Second Edition* (pp. 37–54) and the newer *SAS Macro Language: Reference*, *First Edition* (pp. 50–66).

Papers that specifically cover referencing environments include Bercov (1993) and Hubbell (1990).

An example of a macro variable that takes on more than one value at the same time is given in Carpenter (1996, p. 1637).

1.5 Chapter Summary

You can think of the macro facility as a part of SAS that passively waits to be evoked. If your SAS code contains no macros and no references to macro variables or macro statements, then the macro facility is not used. When the code does contain macro language references, the macro facility wakes up, intercepts the job stream, interprets or executes the macro references, and then releases its control.

The macro facility is made up of two primary components. The *macro processor* provides the ability to compile and execute the *macro language* statements that you use to write macros.