

Paper 120-2013

Macro Basics for New SAS® Users

Cynthia L. Zender, SAS Institute Inc., Cary, NC

ABSTRACT

Are you new to SAS®? Do you look at programs written by others and wonder what those & and % signs mean? Are you reluctant to change code that has macro variables in the program? Do you need to perform repetitive programming tasks and don't know when to use DO versus %DO?

This paper provides an overview of how the SAS macro facility works and how you can make it work in your programs. Concrete examples answer these and other questions: Where do the macro variables live? What does it mean when I see multiple ampersands (&)? What is a macro program, and how does it differ from other SAS programs? What's the big difference between DO and IF and %DO and %IF?

INTRODUCTION

When I think of the SAS® macro facility, I think of two different things. I think of the find-and-replace feature that you can find in almost every computer application, and I think of old-time typists sitting in a room and transcribing dictation (think of *Mad Men* and the secretaries who take dictation, but without the hanky-panky). So, you might wonder what any of that has to do with macro basics because you thought I was going to talk about Excel macros, right? Wrong!

First, let's start with Dictionary.com:

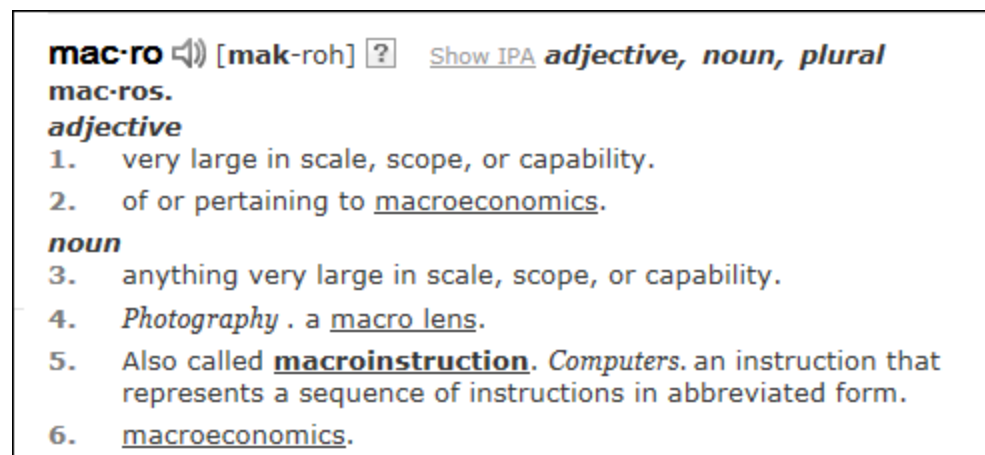


Figure 1: Dictionary.com Entry for the Term *Macro*

Of these six possibilities for meaning, we can eliminate numbers 2, 4 and 6. That leaves number 1 and number 3, which are virtually the same thing: a macro is very large in scale, scope, or capability, which is true of the macro facility. But number 5 comes closest of all to a description of the essence of the macro facility: "an instruction that represents a sequence of instructions in abbreviated form."

According to Wikipedia (http://en.wikipedia.org/wiki/Macro_instruction), the concept of macro instructions goes all the way back to assembly language. With assembly language programs (back in the 1950s), macro instructions were used for two purposes: code reduction and code standardization. Macro instructions would be processed by a macro compiler and the end result was one or more assembly language instructions that comprised the final code used to accomplish a task or operation. This concept still holds true for the macro facility. It works on three principles: abbreviation, substitution, and expansion (also known as code generation).

WHAT IS THE SAS MACRO FACILITY?

The SAS macro facility allows you to write your SAS programs in such a way that when you specify abbreviations in your code (or when SAS Enterprise Guide® inserts abbreviations in your code), the macro processor performs substitution and code expansion, based on your instructions. Your final code, without any abbreviations, is sent to be executed, after all the substitution and expansion is finished.

The SAS macro language is how you communicate with the SAS macro processor. Just as the typist on *Mad Men*

sits between the ad agency executive and the final document, the SAS macro processor sits between you and SAS. Every time SAS code is submitted, the code is scanned for macro abbreviations that the macro processor expands, using find-and-replace, and generates SAS code, which is the final code that is compiled and executed.

HOW DOES THE MACRO FACILITY WORK?

The macro facility has two major components: the macro language and the macro processor. Special abbreviations in your code trigger the macro processor to do something. The special abbreviations are the & and % symbols, which are called macro *triggers*. The & followed by a name (&macvar) is called a macro *variable*. The &macvar reference causes the macro processor to do the equivalent of find-and-replace. The % followed by a name (%macname) can call a predefined macro (either a macro program, a macro function, or a macro statement). The macro functions or macro programs instruct the macro processor to perform a more elaborate find-and-replace with more code expansion and code generation than you can get with simple macro variable references.

By the time this paper is over, you will see concrete examples of using SAS macro variables and SAS macro programs. And, you'll take away a baker's dozen of basic tips and essential facts that will begin your introduction into the many powerful features of the macro facility. But, before we get into examples with real SAS code or Enterprise Guide tasks, let's start with a simple find-and-replace example.

WORKING WITH FIND-AND-REPLACE

Consider this partially finished sentence with keyboard characters as my abbreviations:

The ~ in # stays ^ on the @.

I want the following substitutions: the ~ should have "rain" substituted; the # should have "Spain" substituted; the ^ should have "mainly" substituted; and the @ should have "plain" substituted, as shown in Figure 2.

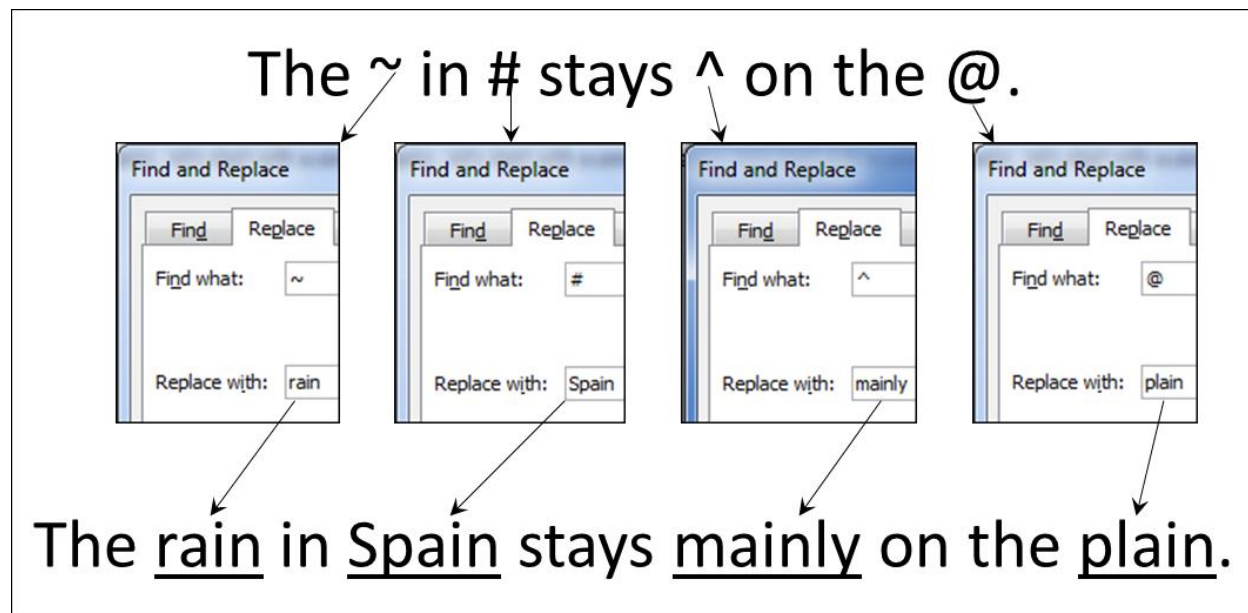


Figure 2: Using Find-and-Replace for Character Substitution

And that's fine if I'm using a word processor or a text editor, but what if there are other substitutions I want to make? What if I want to generate any of these three sentences:

The rain in Spain stays mainly on the plain.

The mess in my house stays mostly on the floor.

The Cat in the Hat stays silly on the pages.

SAS macro variables to the rescue! If I decide on the names I want to use for my macro abbreviations, or macro

variables, then I can write my code like this:

The &what in &place stays &how on the &where.

Now, the macro variables &WHAT, &PLACE, &HOW, and &WHERE will trigger substitution behavior in the macro processor, if submitted inside a SAS program. The simplest way to see how macro variables will resolve is to use them in a %PUT statement. But when I code a %PUT statement with my sentence, I see a problem in the SAS log:

```
398 %put The &what in &place stays &how on the &where.;
WARNING: Apparent symbolic reference WHAT not resolved.
WARNING: Apparent symbolic reference PLACE not resolved.
WARNING: Apparent symbolic reference HOW not resolved.
WARNING: Apparent symbolic reference WHERE not resolved.
The &what in &place stays &how on the &where.
```

Figure 3: Warning Messages in the SAS Log

The SAS macro processor tried to take my abbreviations and do substitution, but the macro processor didn't know the proper values to substitute for &WHAT, &PLACE, &HOW, and &WHERE, so the %PUT statement merely wrote the unresolved text string into the SAS log, after warning me that replacement (or resolution) could not be performed.

What happens behind the scenes is that when I submit the %PUT statement, the code actually is scanned in order to find out whether there are any macro triggers. If macro triggers are found, the macro processor has to go look up &MACVAR references in a special place called the global symbol table. If I had written a different %PUT statement, one that used macro variables that SAS created at start-up time, I would have gotten different results:

```
744 %put The date is: &sysdate9;
The date is: 23DEC2012
```

Figure 4: Using an Automatic SAS Macro Variable

The SAS macro variable &SYSDATE9 is a global macro variable that is automatically created when a SAS session starts. It lives in a table in memory and you can always see the list of all the global macro variables used in your SAS session by issuing another very useful %PUT statement:

```
%PUT _AUTOMATIC_;
```

The partial log results from this code are shown in Figure 5.

```
745 ** Write the automatic global macro variables to the Log;
746 %PUT _AUTOMATIC_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSADDBITS 64
AUTOMATIC SYSBUFFER
AUTOMATIC SYSCC 3000
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 23DEC12
AUTOMATIC SYSDATE9 23DEC2012
AUTOMATIC SYSDAY Sunday
```

Figure 5: Partial List of Automatic Macro Variables

There are other %PUT statements we can use to reveal other global macro variables, but the ones I'm interested in (&WHAT, &PLACE, &HOW, and &WHERE) all need to have values assigned. Next, we'll see how to make that happen.

GETTING STARTED WITH SAS MACRO VARIABLES

There is more than one way to assign values to macro variables but, right now, the easiest way to assign values to the four macro variables that I want to use is with a %LET statement.

The %LET statement allows a programmer to create his or her own user-defined macro variables and have them placed in the global symbol table for the duration of a SAS session. This means that any time the macro variable is referenced inside a session, the macro processor can do find-and-replace substitution on the text. I think of this as having the macro processor type my programs (or pieces of my programs) for me. So, how do I use a %LET statement to generate my desired sentence? To assign a value to a user-defined macro variable, use a simple assignment statement:

```
%let macvar = value;
```

where %LET is the macro statement that starts the assignment,

macvar is the name of the macro variable that you want to create, and

value is the value you want to have entered in the Global Symbol Table.

Here is the revised program:

```
** Use %LET to create macro variables.;
%let what = rain;
%let place = Spain;
%let how = mainly;
%let where = plain;
%put The &what in &place stays &how on the &where.;
```

The results in the SAS log are shown in Figure 6.

```
754
755 ** Use %LET to create macro variables.;
756 %let what = rain;
757 %let place = Spain;
758 %let how = mainly;
759 %let where = plain;
760
761 %put The &what in &place stays &how on the &where.;
The rain in Spain stays mainly on the plain
```

Figure 6: Log Results

If you carefully compare the %PUT statement to the resolved text in the SAS log, you will notice a difference: the period at the end of the sentence has disappeared!

This is one of the important features of the macro facility. Periods have special syntactical meaning to the macro processor. They are used to signal the end of a macro variable reference, so if any macro variable is immediately followed with a period, the period is considered to be a delimiter and is not part of the resolved macro variable value. So, if I had code like this:

```
%let front = c;
%let back = andy;
%put ====> &front.&back;

%let front=d;
%put ====> &front.&back;
```

The results are shown in Figure 7.

```
767 %let front = c;
768 %let back = andy;
769 %put ====> &front.&back;
====> candy
770
771 %let front=d;
772 %put ====> &front.&back;
====> dandy
```

Figure 7: Log Results

Notice how the "c" or the "d" is used in front of "andy", without any period. In effect when the macro processor did find-and-replace, it replaced the &FRONT macro variable because the reference &FRONT. (with the period) meant something special to the macro processor.

In truth, the reference &FRONT&BACK would also have worked with text strings, as shown in Figure 8.

```
1007 %let front=h;
1008 %let back = andy;
1009 %put ====> &front&back;
====> handy
```

Figure 8: Log Results

But if I did need to have a period in my generated text string, I would have to use the double-dot technique to get it, as shown in Figure 9.

```
1018 %let front=The dataset is SASHELP;
1019 %let back=CLASS;
1020 %put &front&back;
The dataset is SASHELPCLASS
1021 %put &front..&back;
The dataset is SASHELP.CLASS
```

Figure 9: Using Double Dots

So, using the same double-dot technique, I can get a period at the end of my sentence. In fact, there's more than one way to generate the period at the end of my sentences. Each of the code examples in Figure 10 shows one method.

```
1026 %let what = mess;
1027 %let place = my house;
1028 %let how = mostly;
1029 %let where = floor;
1030
1031 %put The &what in &place stays &how on the &where..;
The mess in my house stays mostly on the floor.
1032
1033
1034 %let what = Cat;
1035 %let place = the Hat;
1036 %let how = silly;
1037 %let where = pages.;
1038
1039 %put The &what in &place stays &how on the &where;
The Cat in the Hat stays silly on the pages.
```

Figure 10: Generating Different Text Strings with Macro Variables

One syntax method to get the period at the end of my sentence is to use the double dots in the %PUT statement and the alternate method is to specify the period for the end of the sentence as part of the macro variable value. Either way will work. For the SASHELP.CLASS example, I would recommend the double-dot method. For punctuation at the end of a text sentence, I would recommend either method. However, the second method is not something that should be used lightly; as a rule of thumb, it is a bad idea to pre-quote or pre-supply syntax punctuation in the value of a macro variable. In this example, the punctuation was truly part of a text sentence and not part of code being generated. The next important feature that this illustrates is that the macro facility is only performing the equivalent of find-and-replace. It is only typing the text, and what happens with the text that's been typed is up to the SAS compiler and whether the text that's been typed is used in something other than a %PUT statement.

DOING MORE WITH SAS MACRO VARIABLES

How about doing something with SAS macro variables that doesn't involve a %PUT statement? One of the first rules of working with SAS macro programs is to start with a working SAS program. Here's a simple program that does a PROC PRINT and PROC CONTENTS on SASHELP.CLASS. The OBS= option limits the number of rows that will be displayed in the final PROC PRINT report. The partial output from the program is shown in Figure 11.

Macro Basics for New SAS Users, continued

```
ods _all_ close;
ods noptitle;
ods html file='c:\temp\report.html'
  style=sasweb;
proc print data=sashelp.class(obs=5);
  title "SASHELP.CLASS: Obs=5";
run;

ods select attributes variables sortedby;
proc contents data=sashelp.class;
  title "Documentation for: SASHELP.CLASS";
run;
ods html close;
```

SASHELP.CLASS: Obs=5					
Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Janet	F	15	62.5	112.5

Documentation for: SASHELP.CLASS			
Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Sunday, November 11, 2012 09:35:41 AM	Observation Length	40
Last Modified	Sunday, November 11, 2012 09:35:41 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		

Figure 11: Partial Output from the Working SAS program

Now that there is a working SAS program, the program can be "macro-ized" by identifying the code elements that could be useful if they were changeable. For example, I might want the flexibility to change the library, the data set, and the number of observations. Other choices could have been made, but for now, these three choices will suffice:

```
ods html file='c:\temp\report.html'
  style=sasweb;
proc print data=&lib..&data(obs=&numobs);
  title "&lib..&data: Obs=&numobs";
```

Macro Basics for New SAS Users, continued

```
run;

ods select attributes variables sortedby;
proc contents data=&lib..&data;
  title "Documentation for: &lib..&data";
run;
ods html close;ods html close;
```

The three macro variables that will be used are &LIB, &DATA, &NUMOBS. The next thing to do is to test the program and make sure that the changed program produces the same results as the original program. That means three %LET statements are needed before the program code is submitted. The output in Figure 12 shows %LET statements that assign values to the macro variable values, and the successful execution of the SAS code as reporting in the SAS log. The HTML results are almost the same as the original file, as shown in Figure 13.

```
4651 %let lib=sashelp;
4652 %let data = class;
4653 %let numobs = 5;
4654
4655 ods html file='c:\temp\report.html'
4656       style=sasweb;
NOTE: Writing HTML Body file: c:\temp\report.html
4657   proc print data=&lib..&data(obs=&numobs);
4658       title "&lib..&data: Obs=&numobs";
4659   run;

NOTE: There were 5 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds

4660
4661   ods select attributes variables sortedby;
4662   proc contents data=&lib..&data;
4663       title "Documentation for: &lib..&data";
4664   run;

NOTE: PROCEDURE CONTENTS used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

4665 ods html close;
```

Figure 12: Program Shown in the SAS Log

As you can see in Figure 13, the report rows are the same, but the SAS title is a bit different from the original report output. The text for &LIB and &DATA were replaced by the macro processor exactly as they were specified in the %LET statements. Instead of being in uppercase, as originally specified, the library and data set names are shown exactly as they were specified in the %LET statement. This should serve to reinforce the message that the macro facility is only typing text by essentially performing find-and-replace in this instance, without any changes to the original text strings.

sashelpclass: Obs=5					
Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Janet	F	15	62.5	112.5

Documentation for: sashelp.class			
Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Sunday, November 11, 2012 09:35:41 AM	Observation Length	40
Last Modified	Sunday, November 11, 2012 09:35:41 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_64		
Encoding	latin1 Western (Windows)		

Figure 13: Partial HTML Results Viewed in the Browser

What was sent to the macro processor is shown on the left of Figure 14, but what was sent forward to the SAS compiler is shown on the right of Figure 14.

What Was Sent to the Macro Processor	The Resolved Text
<pre>ods html file='c:\temp\report.html' style=sasweb; proc print data=&lib.&data (obs=&numobs); title "&lib.&data: Obs=&numobs"; run; ods select attributes variables sortedby; proc contents data=&lib.&data; title "Documentation for: &lib.&data"; run; ods html close;</pre>	<pre>ods html file='c:\temp\report.html' style=sasweb; proc print data=sashelp.class (obs=5); title "sashelp.class: Obs=5"; run; ods select attributes variables sortedby; proc contents data=sashelp.class; title "Documentation for: sashelp.class"; run; ods html close;</pre>

Figure 14: What the Macro Processor Got Versus What the SAS Compiler Got

As you can see, all the macro variable references were replaced with other text by the time the program was sent to the compiler for syntax checking. If I want the library and data set name to be capitalized in the SAS title, or I want

them to appear in mixed case, then I have to use other special features of the macro facility.

By itself, the macro processor will not alter the text that is stored with a %LET statement. However, I need the values to be changed inside the macro processor before the replaced text is sent to the compiler. Luckily, the macro facility has the %UPCASE function, for changing the case of a macro variable to uppercase. This macro function works just like the programming language UPCASE function, but the % before the UPCASE makes the function a macro function and the % will trigger the function to work inside the macro processor. So I can use this and other macro functions for a special type of find-and-replace. When I use %UPCASE, all the characters will be replaced with their uppercase equivalents. But I haven't decided whether I really want uppercase for the data set name or mixed case. I want to see the PROC CONTENTS title in mixed case, so I can decide.

There is not a macro version of every programming language function. Instead, the SAS macro language has a way for you to invoke many of the programming language functions with a special macro instruction called %SYSFUNC. Since I know that there is a programming language function that will only capitalize the first letter of a text string or variable value, I can use the PROPCASE function with %SYSFUNC to get what I want in the SAS title. Here's an example of using %UPCASE and %SYSFUNC:

```
%put =====> %upcase(abcdefg123 456hijklmn789);
%put =====> %sysfunc(propcase(every first letter will be upcase));
```

The results are shown in Figure 15.

```
1809 %put =====> %upcase(abcdefg123 456hijklmn789);
=====> ABCDEFG123 456HIJKLMN789
1810 %put =====> %sysfunc(propcase(every first letter will be upcase));
=====> Every First Letter Will Be Upcase
```

Figure 15: Using %UPCASE and %SYSFUNC

To get the titles I want, I only have to modify the TITLE statements from the previous code to the TITLE statements shown in Figure 16. Figure 16 shows both the changed TITLE statement and the results.

Notice in Figure 16 that the only quotation marks used are the quotation marks that belong to the TITLE statement. This is because the macro facility knows that it is only doing find-and-replace of text strings, so it doesn't need to have the arguments to these functions quoted the way they would need to be quoted if they were text strings in a SAS program.

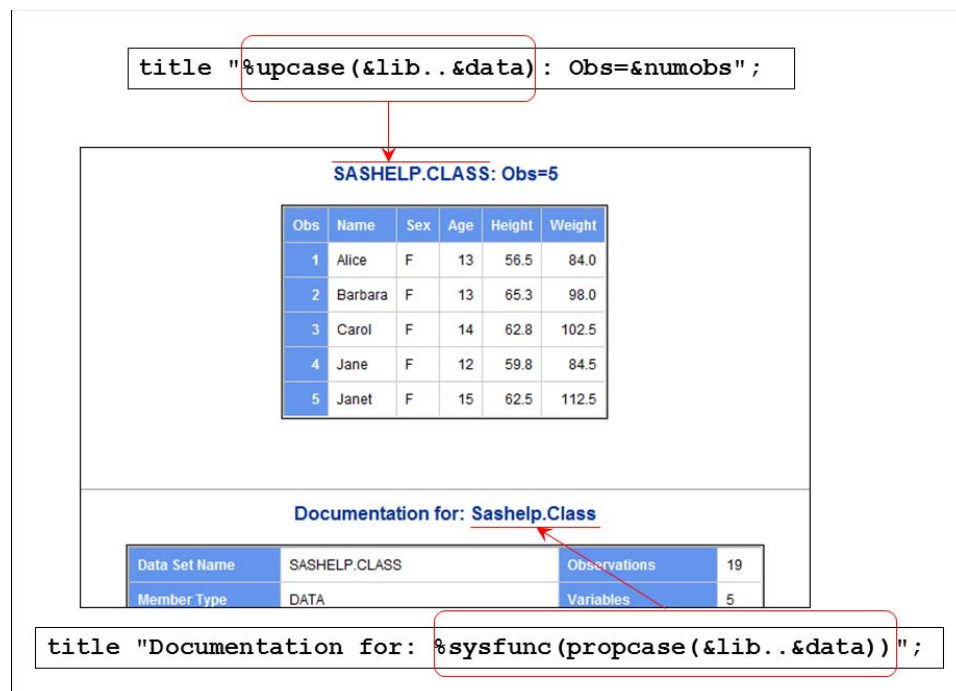


Figure 16: Changed TITLE Statement and Results

This is another important feature of the macro facility: it is only a text generator. The text could be pieces of code, whole steps, or whole programs, but you have to understand what text you want to have generated in order to code your macro functions correctly. And normally, in SAS syntax, whether you use single or double quotation marks doesn't matter. But look what happens in Figure 17 if I use single quotation marks for the TITLE statement in the PROC PRINT step.

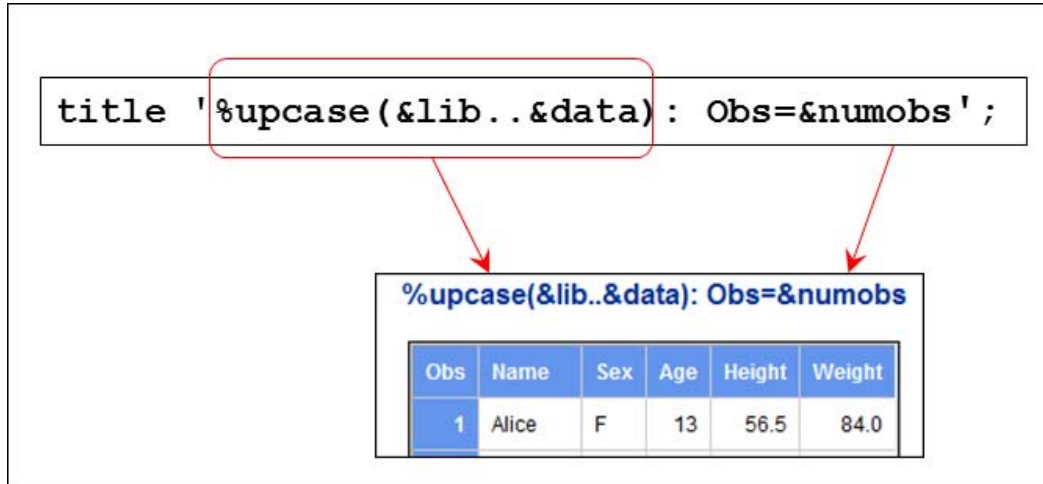


Figure 17: Using Single Quotation Marks with Macro Variable References

The single quotation marks provide a way for the macro triggers of % and & in the title to be protected from the find-and-replace action of the macro processor. The unresolved text string is protected and passed, unchanged, to the compiler for use as the report title. The bottom line is that if you want your % and & to be recognized as macro triggers, and you want find-and-replace to happen, then use double quotation marks where you need to have quotation marks in your syntax.

So far, the important basic points about the macro facility are these:

1. The & and % are macro triggers to cause the find-and-replace of text strings in the macro processor. Usually, you will see multiple ampersands && in code written by other people. You rarely see multiple percent signs %% (although it is possible). The use of multiple ampersands represents a macro feature called *indirect reference*. This is just a way to use one macro variable to help you build the reference to another macro variable. A simple example of indirect reference is shown in Figure 18. A longer explanation of indirect reference and the rules that control multiple ampersand resolution would take this paper significantly beyond the allotted page limit. I recommend that you look in the SAS macro documentation for some good examples.
2. The %LET statement is one way of creating user-defined macro variables. And the %PUT statement is one way to detect how macro variable references will be resolved.
3. Dots (periods) make a difference in macro variable references.
4. The macro processor only does find-and-replace of text, essentially doing typing for you. The macro processor doesn't care whether it types characters or numbers. It is only typing.
5. There are macro functions that you can use to alter the appearance of your resolved macro variable references.
6. If your SAS code needs quotation marks (such as a TITLE or FOOTNOTE statement) and you use macro variable references inside the quoted code, then use double quotation marks if you want the macro variable references resolved. No find-and-replace is done if the macro references are enclosed in single quotation marks.

The resolution process of multiple ampersand references allows you to use a macro variable such as &NUM in several ways, as shown in Figure 18. The simple reference Muppet&NUM is resolved based on the macro processor doing its scanning from left to right. If the value of &NUM is 1, then Muppet&NUM resolves to the string Muppet1. With the reference &&muppet&NUM, the two ampersands at the front of the macro variable reference cause a delay on the first scan so that &NUM can be resolved first and turned into the number 1. Then the delayed resolution happens on a second scan because what the macro processor is holding is &muppet1, which resolves to the string, Kermit. Later in the paper we will talk about an option that helps you debug all your macro variable references.

```

138 **5) Example of indirect reference;
139 %let muppet1 = Kermit;
140 %let muppet2 = Elmo;
141 %put Muppet1 is &muppet1 and Muppet2 is &muppet2;
Muppet1 is Kermit and Muppet2 is Elmo
142
143 %let num=1;
144 %put I like Muppet&num who is &&muppet&num;
I like Muppet1 who is Kermit
145
146 %let num=2;
147 %put I also like Muppet&num who is &&muppet&num;
I also like Muppet2 who is Elmo

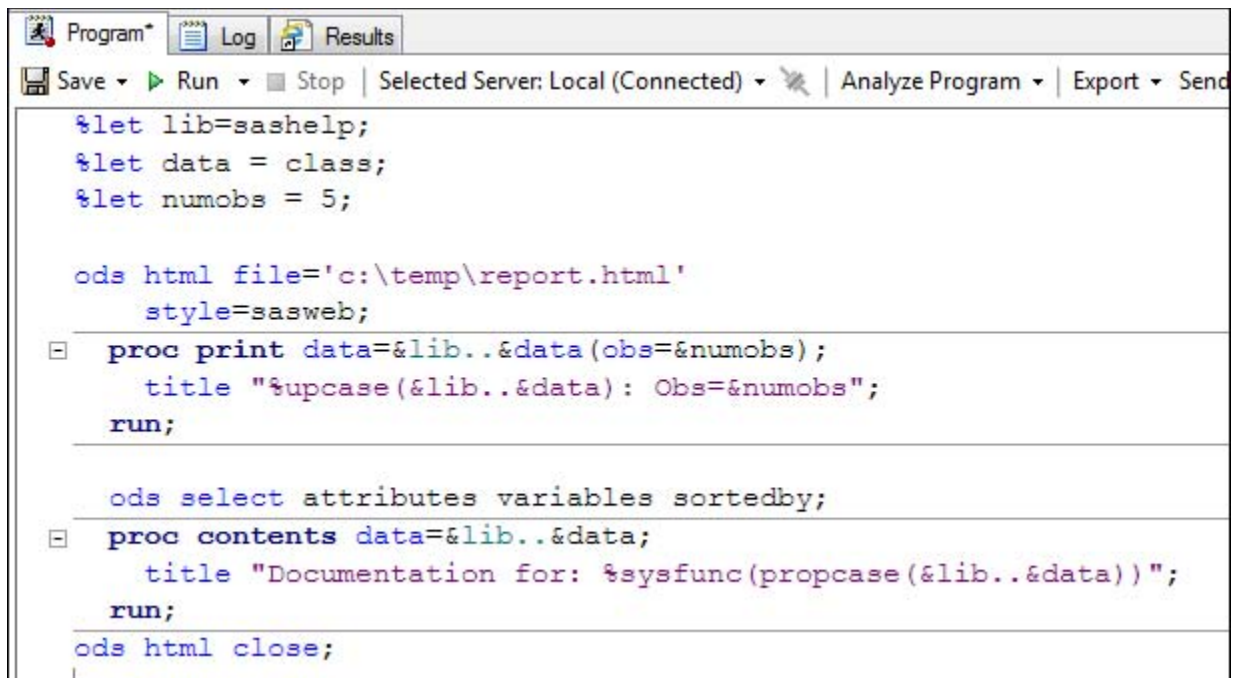
```

Figure 18: Example of Indirect Reference in Macro Variable Usage

So far, all the code has been submitted from SAS windowing environment, although we could have submitted the same code from within Enterprise Guide or we could have added macro variable references to Enterprise Guide tasks and projects. Enterprise Guide does bring something extra to the macro party. So let's take our code over to Enterprise Guide for a look.

USING SAS ENTERPRISE GUIDE:

Moving the code to an Enterprise Guide program window is shown in Figure 19.



```

Program*  Log  Results
Save Run Stop | Selected Server: Local (Connected) | Analyze Program | Export | Send

%let lib=sashelp;
%let data = class;
%let numobs = 5;

ods html file='c:\temp\report.html'
style=sasweb;
proc print data=&lib.&data (obs=&numobs);
title "%upcase(&lib.&data): Obs=&numobs";
run;

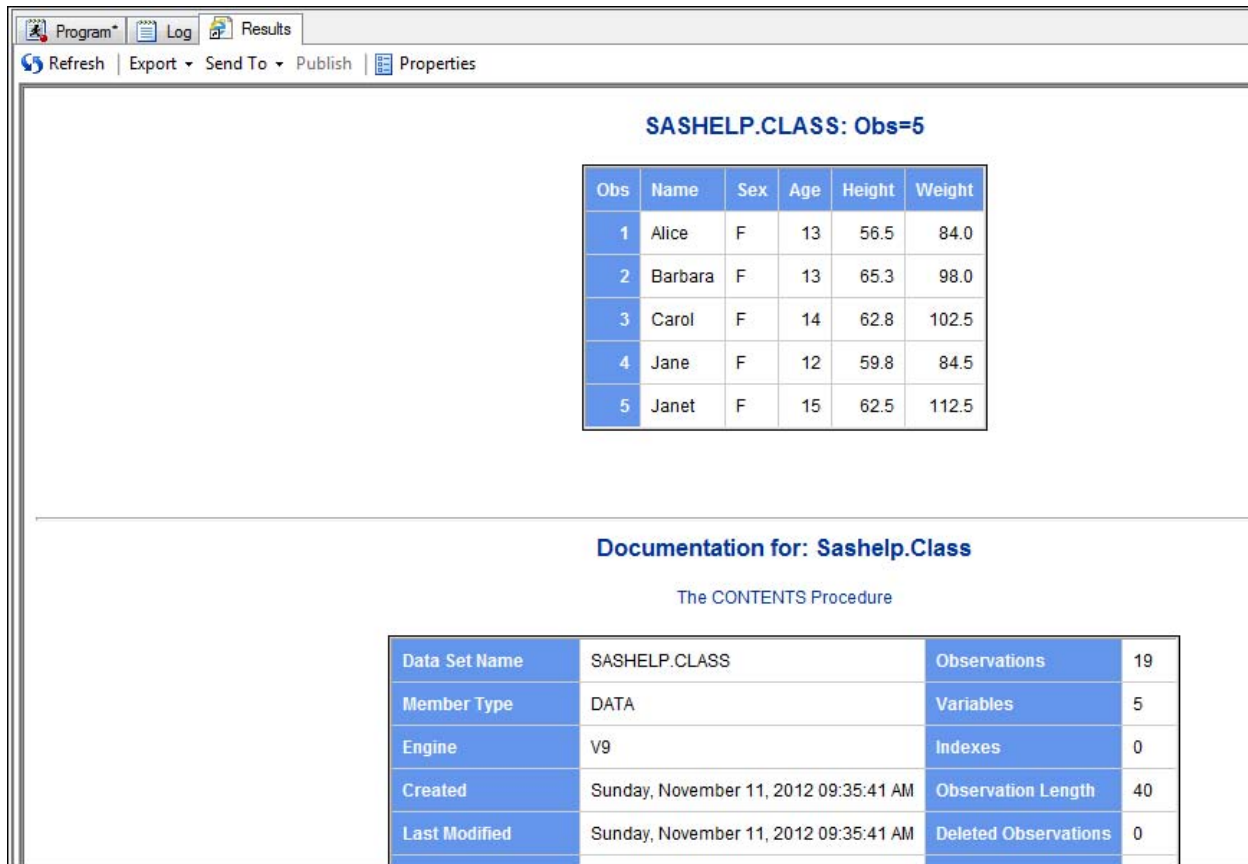
ods select attributes variables sortedby;
proc contents data=&lib.&data;
title "Documentation for: %sysfunc(propcase(&lib.&data))";
run;
ods html close;

```

Figure 19: Code in the Enterprise Guide Program Window

This is the same code that was run and shown in Figure 17. And the results are shown in Figure 20. Enterprise Guide by default creates SAS® Report Model XML output results, so prior to running this code, the **Tools → Options → Results** setting was changed to deselect SAS report output so that the ODS statements controlled the output creation.

Macro Basics for New SAS Users, continued



SASHELP.CLASS: Obs=5

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Janet	F	15	62.5	112.5

Documentation for: SasHELP.Class

The CONTENTS Procedure

Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Sunday, November 11, 2012 09:35:41 AM	Observation Length	40
Last Modified	Sunday, November 11, 2012 09:35:41 AM	Deleted Observations	0

Figure 20: Partial HTML Results Viewed in Enterprise Guide

When you use Enterprise Guide, you have the option to create a type of object called a *prompt*. A prompt is a way to ask for information from the person who runs the task, program, or project. And interestingly enough, a prompt is nothing more than a SAS macro variable that works exactly like the macro variables we have already seen. But Enterprise Guide has some intelligence built in, so that it builds the prompt window automatically from the macro variables you define. To define macro variables to be used as prompts, you click the **Properties** menu on the Program tab, as shown in Figure 21.

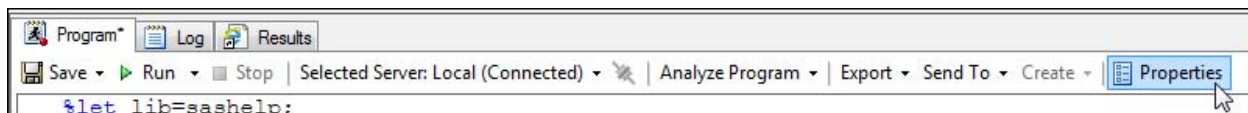


Figure 21: Properties Menu Item

When the Properties window opens, you need to select the **Prompt Manager**, as shown in Figure 22.

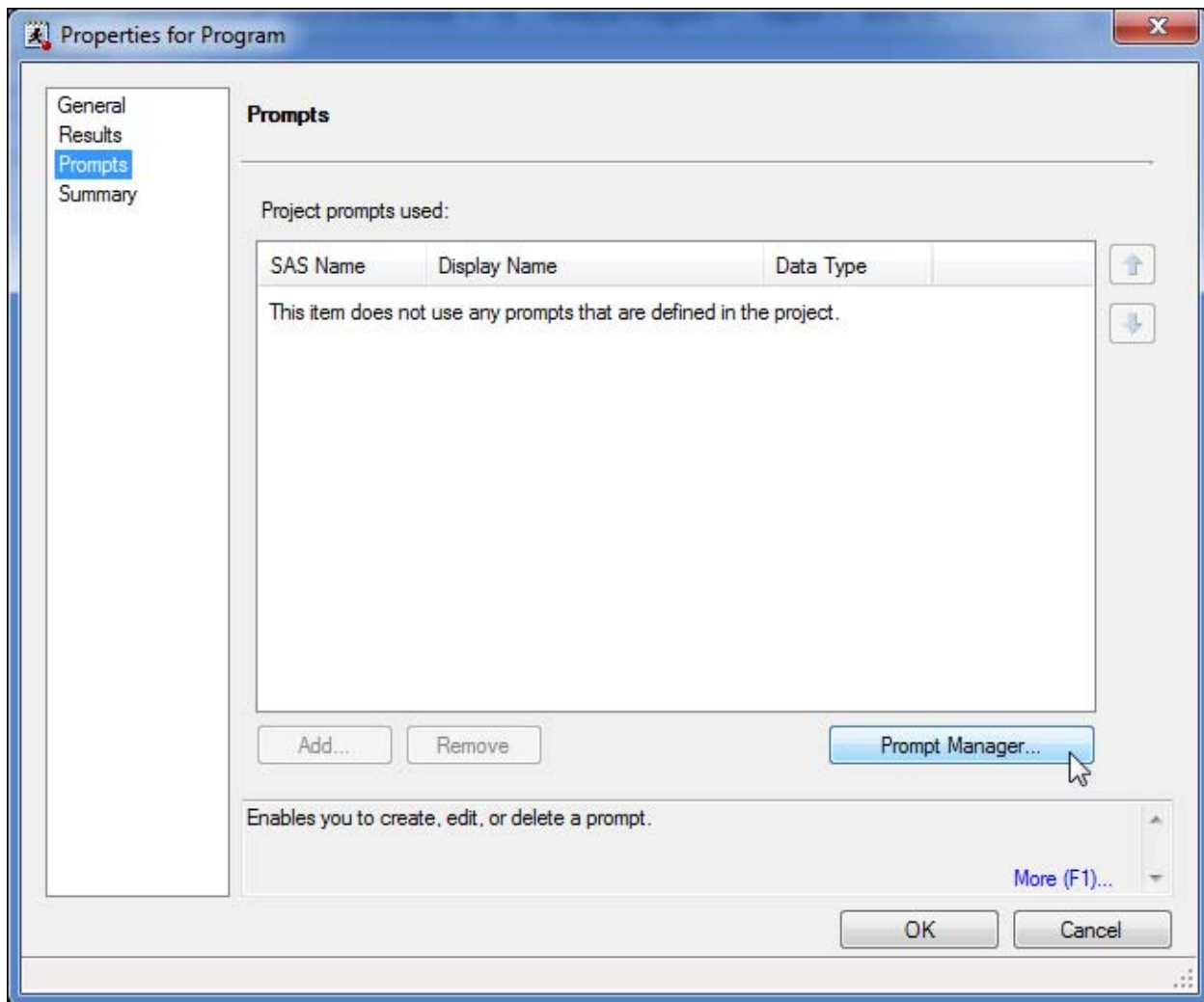


Figure 22: Opening the Prompt Manager

Once you open the Prompt Manager, you click **Add** to add a new prompt. We have three macro variables in the code, and thus we will define three prompts. The nice thing about the Prompt Manager interface is that you can define a default value for the prompt and specify the text that the end user will see.

Figure 23 shows the Prompt Manager window after defining the three macro variables and Figure 24 shows the three macro variables added to the project. Once the hardcoded %LET statements are removed, when the user clicks **Run**, the Enterprise Guide interface will use the Prompt Manager information to build a prompting window as shown in Figure 24. Notice that there are %LET statements in Figure 23, because I use them to remind me what macro variables I use in my code. But, also note how the hardcoded %LET statements have been removed before the final submission of the code. This is necessary because I want the prompt interface values to be used in the macro processor's find-and-replace. I do NOT want the %LET statement values to be used.

Then the user will click **Run** again to send their prompt choices to the macro processor for substitution in the submitted code. Figure 25 shows the partial HTML results in Enterprise Guide, but this time using SASHELP.SHOES and the default value of 5 for the &NUMOBS macro variable.

Macro Basics for New SAS Users, continued

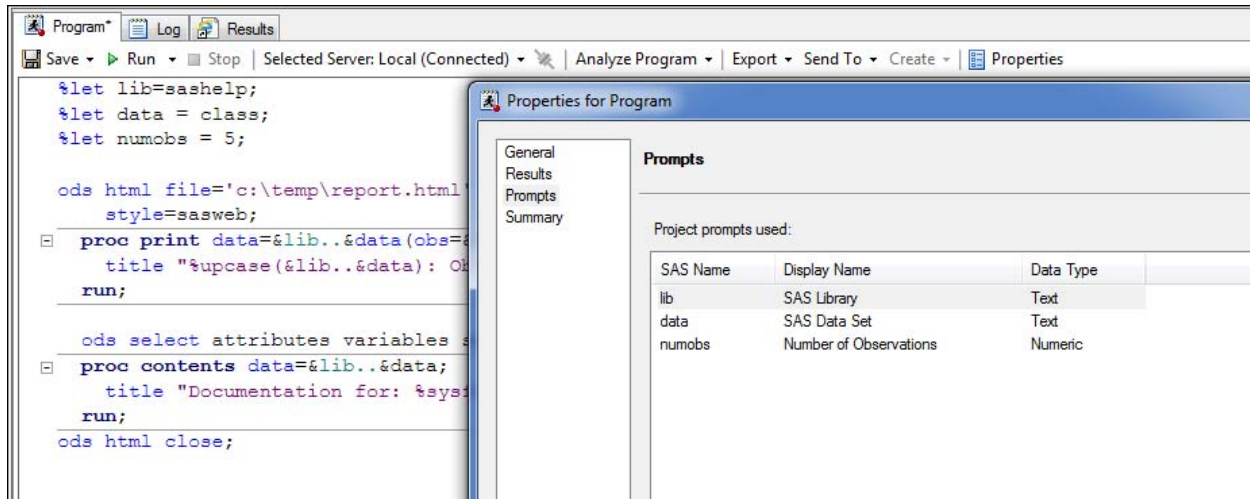


Figure 23: After Defining the Macro Variables in the Prompt Manager

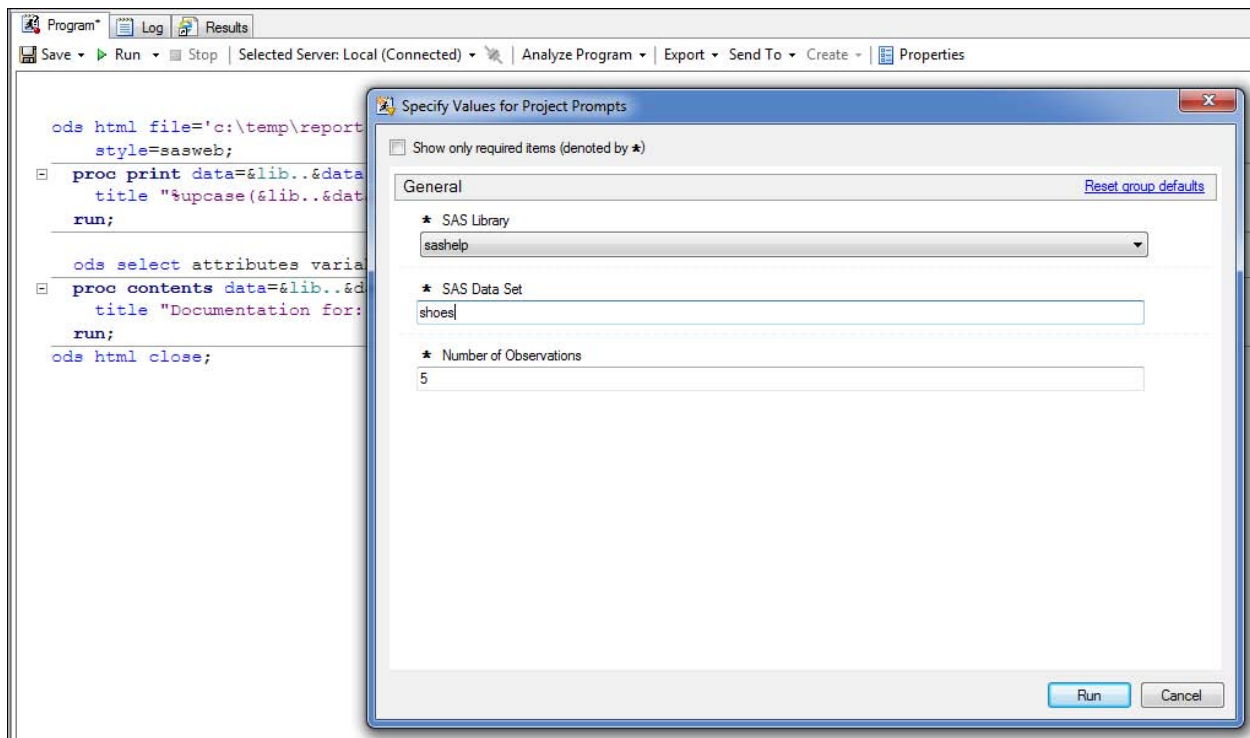


Figure 24: The Automatic Prompt Interface with User Choices

SASHELP.SHOES: Obs=5							
Obs	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
2	Africa	Boot	Algiers	21	\$21,297	\$73,737	\$710
3	Africa	Boot	Cairo	20	\$4,846	\$18,965	\$229
4	Africa	Boot	Johannesburg	14	\$8,365	\$33,011	\$483
5	Africa	Boot	Khartoum	24	\$19,282	\$105,370	\$700

Documentation for: Sashelp.Shoes			
The CONTENTS Procedure			
Data Set Name	SASHELP.SHOES	Observations	395
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	Monday, October 22, 2012 04:22:19 PM	Observation Length	88

Figure 25: Partial HTML Results

The ability to add prompts or use the prompting interface is not the main focus of this paper. This feature of Enterprise Guide allows you to leverage code that uses macro variables in order to deliver model programs that can be used by different people for different reporting purposes. If you are using Enterprise Guide to create stored processes and you are also using the SAS® Enterprise BI Server metadata facility, then this ability to use macro variables and collect prompt information expands beyond Enterprise Guide to all the other client applications on the SAS® Business Intelligence platform.

For a longer discussion of using the Enterprise Guide prompting interface, refer to Sucher (2010)..

However, when you add what you can do with the macro facility in program code to what you can do with the Enterprise Guide Prompt Manager, you get a lot of "bang for your buck." Since we are about to move up to the next layer of SAS macro basics, perhaps the ability to use the prompting interface will help motivate you for the transition from merely using macro variables to writing your own macro programs.

WHAT IS A MACRO PROGRAM DEFINITION?

You've already seen how the macro facility does find-and-replace when you submit code and the macro processor detects and handles the & and the % in the code.

What if you not only want to be able to allow users to select a different library and data set for the program, but you also want them to have the ability to choose to get the PROC PRINT output by itself, the PROC CONTENTS output by itself, or both outputs? And if they ask for a particular data set (SASHELP.CLASS), you want to assign a value of 19 to &NUMOBS.

Normally, in a regular DATA step program, we can use conditional logic to perform tasks based on some condition being true. If I want to project the heights of the students in SASHELP.CLASS, I might want to apply one factor for male students and a different factor for female students. Here's a simple program with some conditional logic:

```
data newheight;
  set sashelp.class; where age ge 13;
  if sex = 'M' then projected_height=height*1.10;
  else if sex = 'F' then projected_height=height*1.075;
run;
```

Macro Basics for New SAS Users, continued

Or, I might want to project both heights and weights with more than one statement being executed if the condition is met:

```
data newhw;
  set sashelp.class;
  where age ge 13;
  if sex = 'M' then do;
    projected_height=height*1.10;
    projected_weight=weight*1.10;
  end;
  else if sex = 'F' then do;
    projected_height=height*1.075;
    projected_weight=weight*1.125;
  end;
run;
```

But, no matter which method I use, every time an observation is read, the DATA step program will evaluate the condition for every observation in order to calculate the height and/or weight using the multiplication factor based on the student's gender. The HTML results from the second data step program are shown in Figure 26.

Projected Heights and Weights for Teenagers						
Sex	Name	Age	Height	Weight	Projected Height	Projected Weight
F	Alice	13	56.5	84	60.74	94.50
	Barbara	13	65.3	98	70.20	110.25
	Carol	14	62.8	102.5	67.51	115.31
	Janet	15	62.5	112.5	67.19	126.56
	Judy	14	64.3	90	69.12	101.25
	Mary	15	66.5	112	71.49	126.00
M	Alfred	14	69	112.5	75.90	123.75
	Henry	14	63.5	102.5	69.85	112.75
	Jeffrey	13	62.5	84	68.75	92.40
	Philip	16	72	150	79.20	165.00
	Ronald	15	67	133	73.70	146.30
	William	15	66.5	112	73.15	123.20

Heights: Boys at 10%. Girls at 7.5%
Weights: Boys at 10%. Girls at 12.5%

Figure 26: Partial HTML Results Showing Results of Conditional Logic

The IF and DO statements belong in the SAS programming language interface. If I want to perform conditional processing in the macro processor, I have to use the SAS macro language equivalent %IF and %DO statements.

Without going into too many syntax details, look what happens when I try to use a simple macro %IF statement in my program. I know that SASHELP.CLASS only has 19 observations, so if the requested data set for the program is

SASHELP.CLASS, I want to set the default value of &NUMOBS to 19. Otherwise, I want the default for &NUMOBS to be 5. You can see the trial code and the resulting error message in Figure 27.

```

685 %let lib=sashelp;
686 %let data = class;
687
688 %if &lib = sashelp and &data = class %then %let numobs = 19;
ERROR: The %IF statement is not valid in open code.
689 %else %let numobs = 5;
ERROR: The %ELSE statement is not valid in open code.
690
691 ods html file='c:\temp\report.html'
692 style=sasweb;
NOTE: Writing HTML Body file: c:\temp\report.html
693 proc print data=&lib..&data(obs=&numobs);
694 title "%upcase(&lib..&data): Obs=&numobs";
695 run;

```

Figure 27: Error Message for Incorrect %IF and %ELSE Usage

I like to think of this error message as the macro processor telling me that "what happens in macro, stays in macro." Or, more bluntly, I can't just send a jumble of macro processing conditional logic to the macro processor mixed in with regular SAS statements (what the error message calls *open code*). Your SAS programs, the statements and procedure steps and ODS statements that you type in a SAS Program Editor window, are essentially open code. You are allowed to use macro variable references and some macro functions in your program open code, as you've seen. However, for more complex find-and-replace scenarios or code generation requests, the macro processor wants to have a neat package of everything you want it to type for you. This *package* is called a *macro program definition*, and your macro language statements (such as %IF or %DO) belong inside this special package of statements.

Here's an example of the type of package or macro program definition that I might use for the program so far:

```

%macro doc_data(lib=sashelp, data=class, numobs=, type=BOTH);
%if &lib = sashelp and &data = class %then %let numobs = 19;
%else %let numobs = 5;

ods html file="c:\temp\report_&data.&type..html"
style=sasweb;
%if &type = P or &type = BOTH %then %do;
proc print data=&lib..&data(obs=&numobs);
title "%upcase(&lib..&data): Obs=&numobs";
title2 "Type of Request is: &type";
run;
%end;
%if &type = C or &type=BOTH %then %do;
ods select attributes variables sortedby;
proc contents data=&lib..&data;
title "Documentation for: %sysfunc(propcase(&lib..&data))";
title2 "Type of Request is: &type";
run;
%end;

ods html close;
%mend doc_data;

```

More is happening in this version of the program than what was shown in Figure 27. First, I want to let the user select either the PROC PRINT or the PROC CONTENTS or both procedures, so a new macro variable &TYPE is introduced into the code. Next, I also want to generate a separate output file every time I use this package. That means instead of using FILE='C:\temp\report.html' as the ODS option to name the output file, I want to use the value of &DATA and the value of the &TYPE macro variable as part of the output filename.

The whole macro package or program definition is enclosed within %MACRO and %MEND statements. It is my habit, although it is not required, to show the name of the macro program (DOC_DATA) in both of these statements. It

makes me feel more organized, especially if I have more than one macro definition in a program or project. In the %MACRO statement, I have included my macro variables in such a way that they become keyword *parameters* as part of the definition. There are a couple of different ways to define parameters (or macro variables) for macro programs. This is my preferred method (and the only one I'm going to show) because I am a control freak. I want to set defaults for my macro variables. If this macro is invoked and the user does not specify a value for &LIB, then the macro processor knows what the default value is. I am still taking control of &NUMOBS at the top of the macro program definition, but I've also added some conditional logic based on values of &TYPE to tell the macro processor what code should be sent forward to the SAS compiler based on the values of my macro variables. Pretty cool!

Of course, Enterprise Guide gives me a way in the Prompt Manager to set the default, too. And it's a good way. Enterprise Guide has a way to do conditional logic at the project level, too. But knowing how to create conditional logic in the macro facility using macro code will only help you when you need to design stored processes from your project code or other programs.

You can make either way work with Enterprise Guide. But for now, I just want to submit the code with the macro definition statements. When I look in the SAS log after the job submission, I don't see any errors, but I don't see anything else, either, as shown in Figure 28.

```

1  ** Example Macro %IF in macro program definition;
2  %macro doc_data(lib=sashelp, data=class, numobs=, type=BOTH);
3
4  %if &lib = sashelp and &data = class %then %let numobs = 19;
5  %else %let numobs = 5;
6
7  ods html file="c:\temp\report_&data.&type..html"
8  style=sasweb;
9
10 %if &type = P or &type = BOTH %then %do;
11   proc print data=&lib..&data(obs=&numobs);
12     title "%upcase(&lib..&data): Obs=&numobs";
13     title2 "Type of Request is: &type";
14   run;
15 %end;
16 %if &type = C or &type=BOTH %then %do;
17   ods select attributes variables sortedby;
18   proc contents data=&lib..&data;
19     title "Documentation for: %sysfunc(propcase(&lib..&data))";
20     title2 "Type of Request is: &type";
21   run;
22 %end;
23
24 ods html close;
25 %mend doc_data;
26
27
```

Figure 28: SAS Log after Submitting %MACRO/%MEND Macro Program Definition

However, a PROC CATALOG step (Figure 29) reveals that something happened. The SAS macro processor compiled the macro, and put the compiled macro package into the WORK.SASMACR catalog. If I had turned on the SAS system option MCOMPILENOTE=ALL, then I would have seen a note in the SAS log, as shown in Figure 30.

Contents of Catalog WORK.SASMACR					
#	Name	Type	Create Date	Modified Date	Description
1	DOC_DATA	MACRO	05Jan13:14:29:51	05Jan13:14:29:51	

Figure 29: PROC CATALOG Results

```

93  %mend doc_data;
NOTE: The macro DOC_DATA completed compilation without errors.
      45 instructions 1364 bytes.
94

```

Figure 30: MCOMPILENOTE=ALL Results in the SAS Log

Well, I've created a package and put it into a WORK catalog, but how do I make the macro processor do something with the package? First I have to invoke the macro program; and in the process of invoking the macro program, I have to provide values for the parameters (macro variables) that I defined. Remember how %UPCASE caused the macro processor to do a special find-and-replace on a macro variable value. I'm going to use the percent sign again, but this time, with the macro program name:

```
%DOC_DATA(. . . specify macro variable values . . .)
```

The % will be the macro trigger and when it is placed in front of the macro program name (%DOC_DATA), I am instructing the macro processor to spring into action, retrieve the DOC_DATA macro program from WORK.SASMACR and start typing code, taking into account what should happen conditionally and what my macro program is constructed to do.

So the macro processor will type some code, exactly as it's written in the macro program, but in other places (where I reference a macro variable), the macro processor will do find-and-replace before it types the code.

Another reason I use keyword macro parameters in my macro program definition is that if I want to use the default (such as LIB=SASHELP), then I don't have to respecify the value for LIB in my invocation. Here are four different invocations for the %DOC_DATA macro program:

```

%doc_data(data=shoes,numobs=10,type=P)
%doc_data(type=P)
%doc_data(data=cars,numobs=7,type=C)
%doc_data(data=orsales,numobs=25)

```

The results from running these four invocations are shown in Figures 31, 32, 33, and 34, respectively. Because the FILE= option was specified as: file="c:\temp\report_&data.&type..html", the four files created were: c:\temp\report_shoesP.html, c:\temp\report_classP.html, c:\temp\report_carsC.html, and c:\temp\report_orsalesBOTH.html for each of the above invocations.

SASHELP.SHOES: Obs=5							
Type of Request is: P							
Obs	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
2	Africa	Boot	Algiers	21	\$21,297	\$73,737	\$710
3	Africa	Boot	Cairo	20	\$4,846	\$18,965	\$229
4	Africa	Boot	Johannesburg	14	\$8,365	\$33,011	\$483
5	Africa	Boot	Khartoum	24	\$19,282	\$105,370	\$700

Figure 31: %doc_data(data=shoes,numobs=10,type=P) Partial Results

Pay careful attention to the invocation, where the &NUMOBS macro variable was specified as 10 (in Figure 31). But why does the output show number of observations as being the number 5? What happened that changed the value that was specified? Remember our original macro program. The code snippet below shows the conditional logic that was inside the top of the macro program definition:

Macro Basics for New SAS Users, continued

```
%if &lib = sashelp and &data = class %then %let numobs = 19;
%else %let numobs = 5;
```

This means that any data set other than SASHELP.CLASS will have the NUMOBS value changed inside the macro processor from whatever was originally specified to 5. It is possible that I might want to issue a note in the log or I might, in production, want to get rid of the %ELSE condition. For the remaining screenshots, however, remember that essentially, &NUMOBS is under the control of the macro facility conditional logic and will be changed inside the macro processor. The SAS log in Figure 35 shows you an example of where the invocation value is being changed to 5 based on the macro logic.

SASHELP.CLASS: Obs=19 Type of Request is: P					
Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Janet	F	15	62.5	112.5
6	Joyce	F	11	51.3	50.5
7	Judy	F	14	64.3	90.0
8	Louise	F	12	56.3	77.0
9	Mary	F	15	66.5	112.0
10	Alfred	M	14	69.0	112.5

Figure 32: %doc_data (type=P) Results

Documentation for: Sashelp.Cars			
Type of Request is: C			
The CONTENTS Procedure			
Data Set Name	SASHELP.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	V9	Indexes	0
Created	Tuesday, May 24, 2011 03:06:30 PM	Observation Length	152
Last Modified	Tuesday, May 24, 2011 03:06:30 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label	2004 Car Data		
Data Representation	WINDOWS_64		
Encoding	us-ascii ASCII (ANSI)		

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		

Figure 33: %doc_data (data=cars, numobs=7, type=C) Partial Results

SASHELP.ORSALES: Obs=5								
Type of Request is: BOTH								
Obs	Year	Quarter	Product_Line	Product_Category	Product_Group	Quantity	Profit	Total_Retail_Price
1	1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.90
2	1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.40
3	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.80
4	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334	7136.80	14337.20
5	1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303	7163.00	12996.20

Documentation for: Sashelp.Orsales			
Type of Request is: BOTH			
The CONTENTS Procedure			
Data Set Name	SASHELP.ORSALES	Observations	912
Member Type	DATA	Variables	8
Engine	V9	Indexes	0
Created	Tuesday, May 24, 2011 02:40:14 PM	Observation Length	112
Last Modified	Tuesday, May 24, 2011 02:40:14 PM	Deleted Observations	0

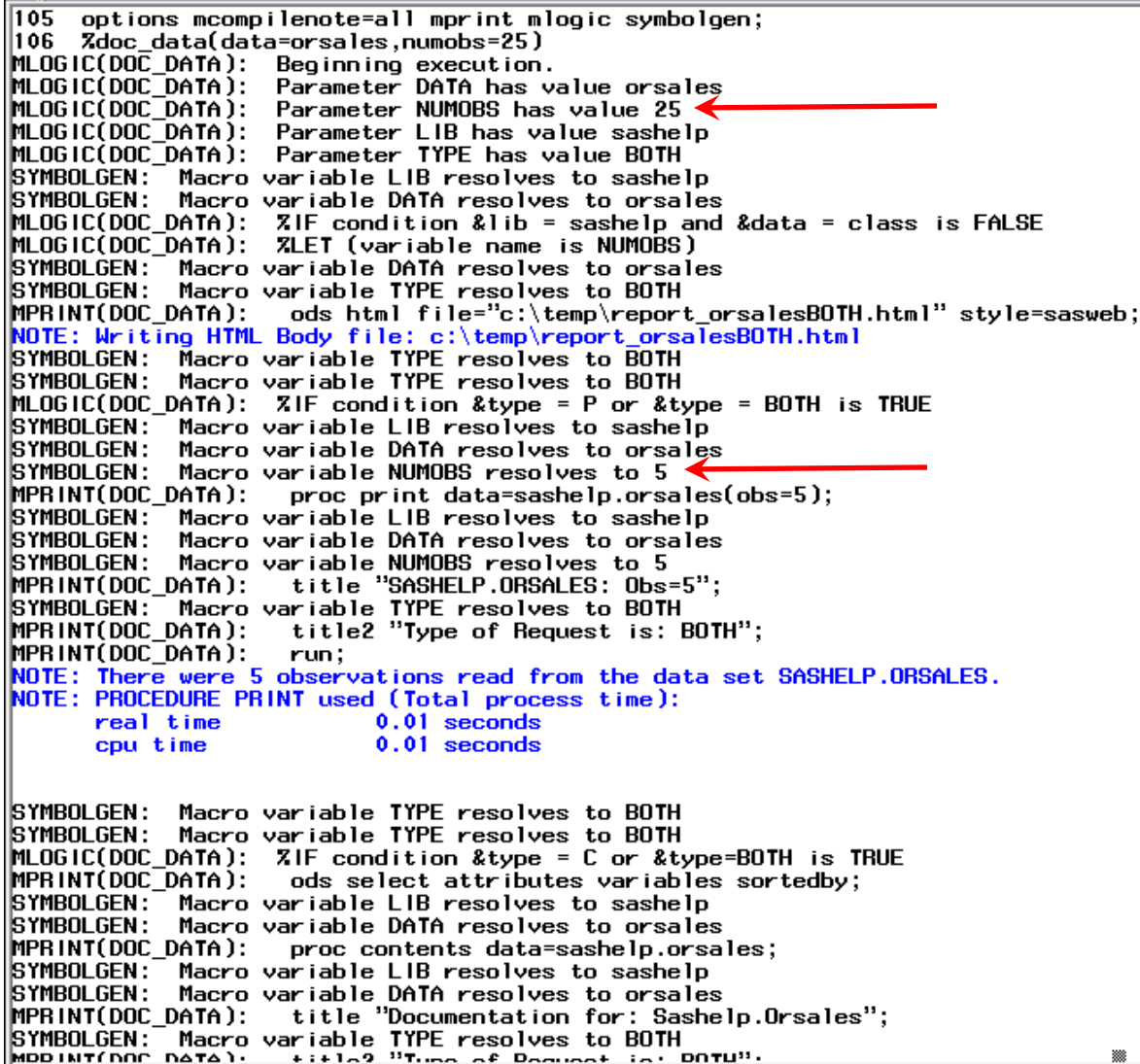
Figure 34: %doc_data (data=orsales, numobs=25) Partial Results

As part of the development process, you can turn on some helpful system options, cousins of MCOMPILENOTE, that will show you how the macro processor resolved the macro variables and macro logic in the code. This set of useful system options, shown with an invocation of our macro program is:

```
options mcompilenote=all mprint mlogic symbolgen;
%doc_data(data=orsales,numobs=25)
```

Partial SAS log results are shown in Figure 35. You can see that every place the macro processor had to do a find-and-replace or had to use conditional logic, the information is echoed in the SAS log. Although this is very useful for development, you will want to turn these options off for production runs:

```
options mcompilenote=none nomprint nomlogic nosymbolgen;
```



```
105 options mcompilenote=all mprint mlogic symbolgen;
106 %doc_data(data=orsales,numobs=25)
MLOGIC(DOC_DATA): Beginning execution.
MLOGIC(DOC_DATA): Parameter DATA has value orsales
MLOGIC(DOC_DATA): Parameter NUMOBS has value 25
MLOGIC(DOC_DATA): Parameter LIB has value sashelp
MLOGIC(DOC_DATA): Parameter TYPE has value BOTH
SYMBOLGEN: Macro variable LIB resolves to sashelp
SYMBOLGEN: Macro variable DATA resolves to orsales
MLOGIC(DOC_DATA): %IF condition &lib = sashelp and &data = class is FALSE
MLOGIC(DOC_DATA): %LET (variable name is NUMOBS)
SYMBOLGEN: Macro variable DATA resolves to orsales
SYMBOLGEN: Macro variable TYPE resolves to BOTH
MPRINT(DOC_DATA): ods html file="c:\temp\report_orsalesBOTH.html" style=sasweb;
NOTE: Writing HTML Body file: c:\temp\report_orsalesBOTH.html
SYMBOLGEN: Macro variable TYPE resolves to BOTH
SYMBOLGEN: Macro variable TYPE resolves to BOTH
MLOGIC(DOC_DATA): %IF condition &type = P or &type = BOTH is TRUE
SYMBOLGEN: Macro variable LIB resolves to sashelp
SYMBOLGEN: Macro variable DATA resolves to orsales
SYMBOLGEN: Macro variable NUMOBS resolves to 5
MPRINT(DOC_DATA): proc print data=sashelp.orsales(obs=5);
SYMBOLGEN: Macro variable LIB resolves to sashelp
SYMBOLGEN: Macro variable DATA resolves to orsales
SYMBOLGEN: Macro variable NUMOBS resolves to 5
MPRINT(DOC_DATA): title "SASHELP.ORSALES: Obs=5";
SYMBOLGEN: Macro variable TYPE resolves to BOTH
MPRINT(DOC_DATA): title2 "Type of Request is: BOTH";
MPRINT(DOC_DATA): run;
NOTE: There were 5 observations read from the data set SASHELP.ORSALES.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

SYMBOLGEN: Macro variable TYPE resolves to BOTH
SYMBOLGEN: Macro variable TYPE resolves to BOTH
MLOGIC(DOC_DATA): %IF condition &type = C or &type=BOTH is TRUE
MPRINT(DOC_DATA): ods select attributes variables sortedby;
SYMBOLGEN: Macro variable LIB resolves to sashelp
SYMBOLGEN: Macro variable DATA resolves to orsales
MPRINT(DOC_DATA): proc contents data=sashelp.orsales;
SYMBOLGEN: Macro variable LIB resolves to sashelp
SYMBOLGEN: Macro variable DATA resolves to orsales
MPRINT(DOC_DATA): title "Documentation for: Sashelp.Orsales";
SYMBOLGEN: Macro variable TYPE resolves to BOTH
MPRINT(DOC_DATA): title2 "Type of Request is: BOTH";
```

Figure 35: Partial Log Results from Macro Program Invocation

In addition to the important macro basics already illustrated, this section has highlighted some of the other basic features of the macro facility (continuing the numbers from the previous list):

7. In order to use SAS macro language statements like %IF and %DO, you have to put your special macro statements in a package called a macro program definition.
8. The %MACRO and %MEND statements delimit the macro program definition.

9. When you submit a macro program definition, by default, it is put into the WORK.SASMACR catalog.
10. To invoke a macro program definition, use the convention **%macro_pgm_name**.
11. If you use parameters, such as keyword parameters, you can specify those values in parentheses as part of the macro invocation.
12. There are system options, such as MCOMPILENOTE, SYMBOLGEN, MPRINT, and MLOGIC, to help you debug your macro program.
13. Unless instructed otherwise (such as with some stored process conventions), SAS macro program invocations do not need to end with a semi-colon.

CONCLUSION

Finally, let's revisit Enterprise Guide to compile, test, and invoke the macro code there, on whatever server Enterprise Guide uses. The program has been changed just a bit. For my macro definition, since I know that the Prompt Manager will hold all my macro variables, and that I am going to define all the macro variables in my project, I am going to use a simpler macro program definition without any keyword parameters.

I know, it seems strange to put the macro variables in the definition for testing and then take them out again. But I know that when I invoke the macro program in Enterprise Guide, the Prompt Manager will provide the necessary values. The same thing is true if I have made a macro program that I was going to use inside a stored process. The prompting interface will provide all the values. You could still use the keyword parameters to define parameters and macro variables that you did not want to surface to your end users, but for now, the code has only been altered a bit. Figure 36 shows the slightly altered code and the final Prompt Manager screen when the user runs the program.

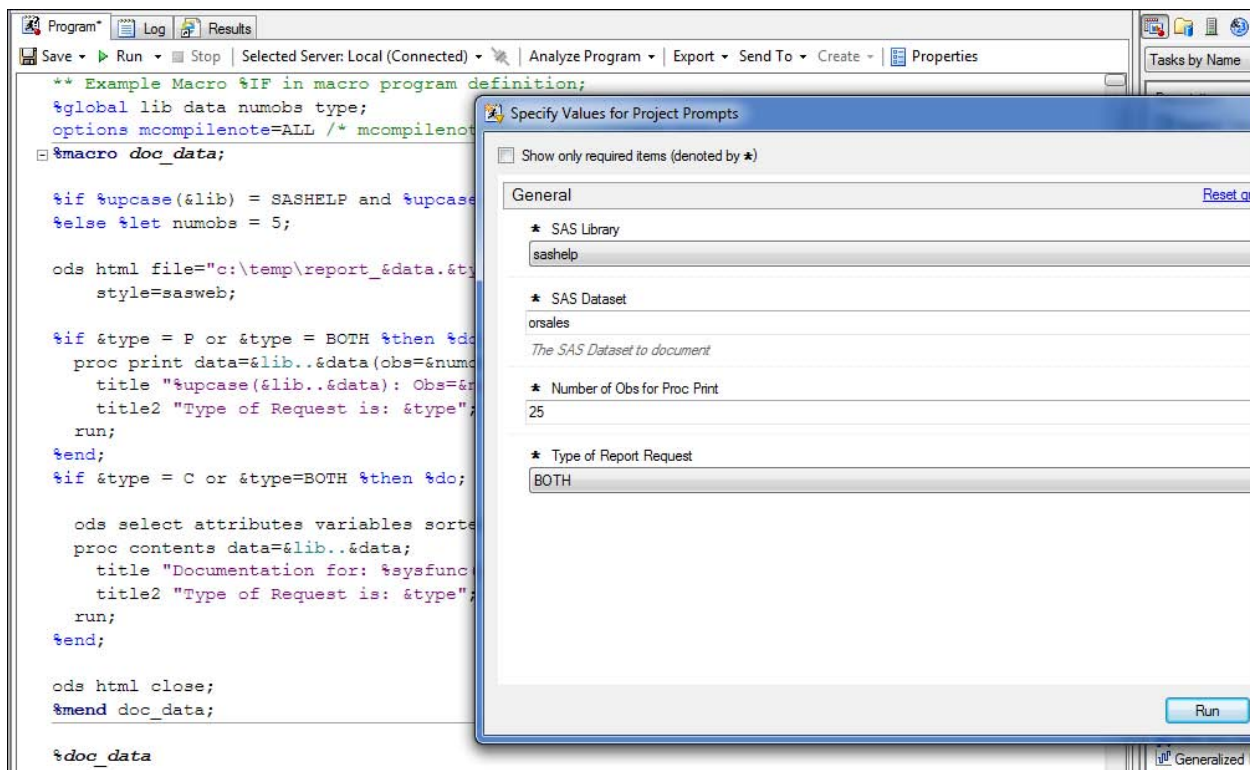


Figure 36: Final Enterprise Guide Program and Prompt Manager Window.

The %GLOBAL statement ensures that the macro processor knows to look for the macro variables from the Prompt Manager in the global symbol table, and the %MACRO statement has been changed to remove keyword parameter definitions. This macro program will be session compiled, so as soon as the macro definition is over, the macro is invoked. This is the simplest way to define and test one macro program. With a more complicated program or more complicated logic, I would probably modularize the single macro program into smaller macro programs, but that is a design discussion that is beyond the scope of this paper.

Once the %DOC_DATA macro program has been compiled, it can be called again without the definition step, as shown in Figure 37. There are ways to define macro programs in an autocall library or from a compiled macro library, to simplify the macro processor's access to the macro program. That also means it is unnecessary for you to compile

the macro each time you want to invoke it. You do have to make sure that the altered program still uses the prompts that were defined for the project.

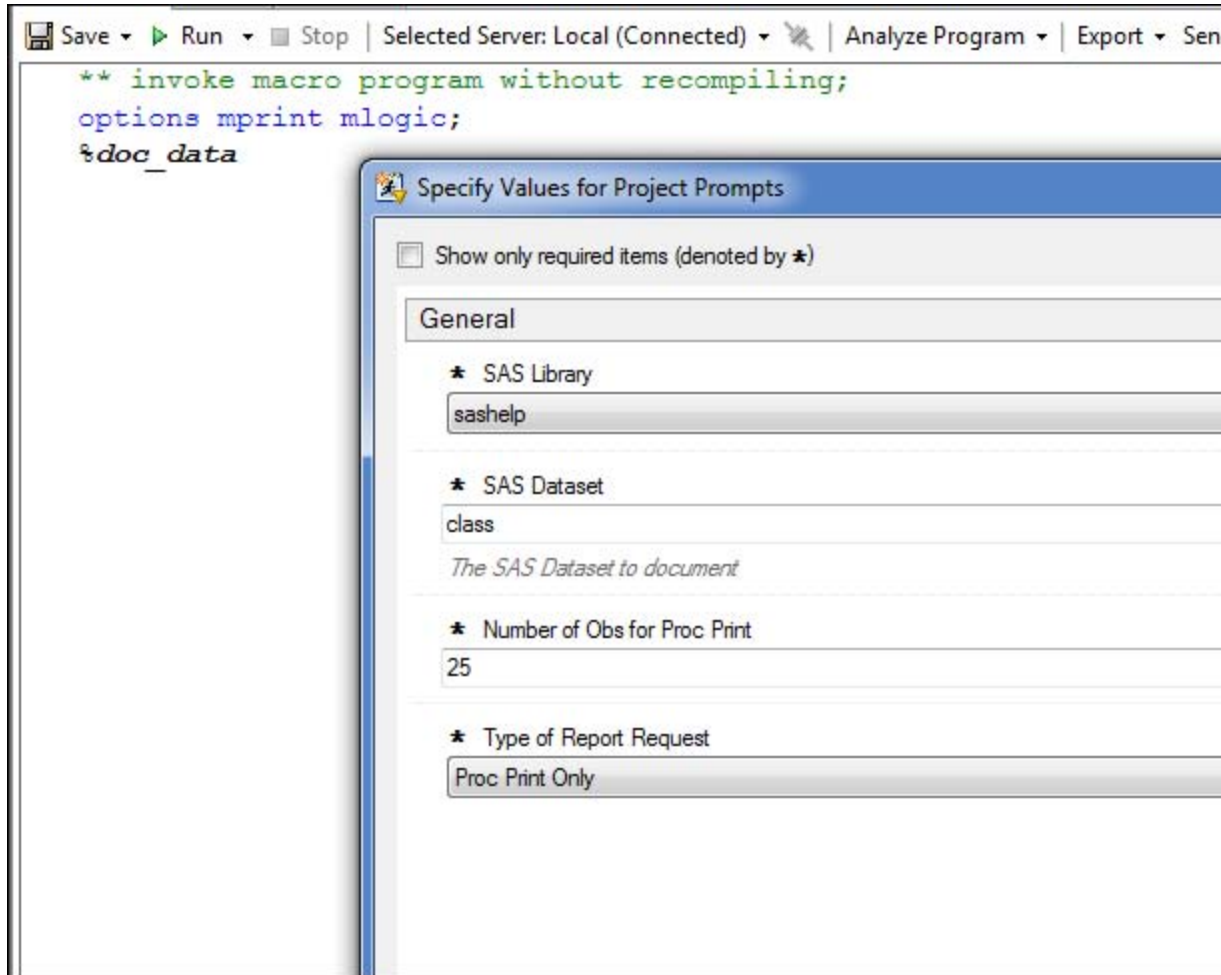


Figure 37: Simple Invocation in Enterprise Guide

Note that the requested data set is SASHELP.CLASS and that the number of observations is set at 25. But remember the macro logic? If the data set is SASHELP.CLASS, there is an %IF statement that resets &NUMOBS to 19. And, sure enough, if we look in the SAS log, as shown in Figure 38, you can see exactly the place where the macro processor acted conditionally and changed &NUMOBS to 19.


```

7      %LET Lib = sashelp;
8      %LET Data = class;
9      %LET NumObs = 25;
10     %LET Type = P;
11
12     ODS _ALL_ CLOSE;
13     OPTIONS DEV=ACTIVEVEX;
14     GOPTIONS XPIXELS=0 YPIXELS=0;
15
16     GOPTIONS ACCESSIBLE;
17     ** invoke macro program without recompiling;
18     options mprint mlogic;
19     %doc_data
MLOGIC(DOC_DATA): Beginning execution.
MLOGIC(DOC_DATA): %IF condition %upcase(&lib) = SASHELP and %upcase(&data) = CLASS is TRUE
MLOGIC(DOC_DATA): %LET (variable name is NUMOBS)
MPRINT(DOC_DATA): ods html file="c:\temp\report_classP.html" style=sasweb;
NOTE: Writing HTML Body file: c:\temp\report_classP.html
MLOGIC(DOC_DATA): %IF condition &type = P or &type = BOTH is TRUE
MPRINT(DOC_DATA): proc print data=sashelp.class(obs=19);
MPRINT(DOC_DATA): title "SASHELP.CLASS: Obs=19";
MPRINT(DOC_DATA): title2 "Type of Request is: P";
MPRINT(DOC_DATA): run;
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.01 seconds
      cpu time           0.00 seconds

MLOGIC(DOC_DATA): %IF condition &type = C or &type=BOTH is FALSE
MPRINT(DOC_DATA): ods html close;
MLOGIC(DOC_DATA): Ending execution.

```

Figure 38: Macro Logic Changes &NUMOBS Value to 19

At the top of the log, &NUMOBS appears in a %LET statement and is set to 25 by the prompting interface. But later, the macro processor encounters the %IF statement and changes NUMOBS to 19, as seen in the resolved, typed, and final PROC PRINT code that was sent to the compiler.

In this paper, we've gone from the very simple concept of find-and-replace to the more complex concept of writing a SAS macro program whose major purpose is to type SAS code conditionally based on input from me or my end users about how the code should be typed behind the scenes. And, we've seen those concepts illustrated in the SAS windowing environment and in Enterprise Guide.

The truth is that there's a lot more to learn about the macro facility and even cooler stuff to learn. My suggestion is that you start slowly, build your confidence, take our excellent SAS macro classes, read some user group papers and experiment. But stick these baker's dozen tips into your programmer's toolkit and remember, "what happens in macro, stays in macro" – the SAS compiler never sees those & or % symbols unless you want it to!

REFERENCES

- Stroupe, Jane. 2003. "Nine Steps to Get Started using SAS Macros." *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at <http://www2.sas.com/proceedings/sugi28/056-28.pdf>.
- Sucher, Kenneth. 2010. "Interactive and Efficient Macro Programming with Prompts in SAS Enterprise Guide 4.2." *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/resources/papers/proceedings10/036-2010.pdf>.
- Burlew, Michele. 2006. *SAS Macro Programming Made Easy*. 2d ed. Cary, NC: SAS Institute Inc. Available at <https://support.sas.com/pubscat/bookdetails.jsp?pc=60560>.
- Carpenter, Art. 2004. *Carpenter's Complete Guide to the SAS Macro Language*. 2d ed. Cary, NC: SAS Institute Inc. Available at <https://support.sas.com/pubscat/bookdetails.jsp?catid=1&pc=59224>.

Macro Basics for New SAS Users, continued

ACKNOWLEDGMENTS

The author would like to Linda Jolley and Jim Simon, whose review comments made this a better paper. Thanks are also due to Maggie Marcum, whose editing skill made this a much better paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Cynthia L. Zender
SAS Institute, Inc.
SAS Campus Drive
919-531-9012
Cynthia.Zender@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.