Advanced Macro Processing
Prepared by Destiny Corporation

Macro Variables vs. Data Step Variables

- Creating Data Step Variables and Macro Variables
- Where the Variables and Values Live

Macro Variables vs. Data Step Variables

It is important to realize that macro variables are not the same as data step variables. Even beginning SAS programmers recognize SAS data set variables. How are macro variables recognized?

- They are created differently from data set variables.
- They are stored in symbol tables, not data sets.
- A macro variable has a single value whereas a data set variable can have multiple values depending on the loop of the data.

Creating Data Set Variables and Macro Variables

SAS can make a data set variable in many ways by using different syntax options within the Data Step.

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00002</td>
<td>The length statement creates space in the PDV for a new data set variable.</td>
</tr>
<tr>
<td>00004-00007</td>
<td>The (list) input statement assigns name, type, and length to new data set variables.</td>
</tr>
<tr>
<td>00009-00010</td>
<td>The variables title1 and title2 are created by assignment.</td>
</tr>
</tbody>
</table>

SAS can also create macro variables in a variety of ways.

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>The %let statement creates macro variable mvar.</td>
</tr>
<tr>
<td>00004</td>
<td>The Call Symput routine creates macro variable day.</td>
</tr>
<tr>
<td>00009</td>
<td>The Proc SQL into clause creates macro variable meanage.</td>
</tr>
</tbody>
</table>

The point to remember: the syntax determines what is being created!

There is no ambiguous case between a data set variable and a macro variable.

Where the Variables and Values Live

Data set variables and macro variables can have the same name.

They will never replace each other’s values, however.

They live in different structures (a symbol table vs. a data set or the PDV) and they are called for use in different manners.
shows macro variable in the Output window.

The Data Set Variable `DAY`

The character data set variable `day` is created through assignment.

The Put statement shows data set variable value in Log.

The Proc Print of data set shows data set value of `day`.

Data Step Programming with Macro Counterparts

Many programming options used in macro bundles have counterparts in SAS data step programming. Review the following series of examples of data step programming. Become familiar with the syntax shown in each.

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00004-00006</td>
<td>The conditional structure.</td>
</tr>
<tr>
<td>00002-00006</td>
<td>The bounded loop structure. Compare <em>do...while</em> and <em>do...until</em>.</td>
</tr>
<tr>
<td>00003-</td>
<td>The <em>goto</em> syntax with label (text) and</td>
</tr>
</tbody>
</table>

Macro Debugging Options

A major challenge of the programmer is to verify that syntax written by a macro and values passed are correct. As for assuring the latter, the programmer can insert a series of `%put` statements in the syntax while it is being written. If the predicted values match the displayed values, the program is in good shape. The SAS session can also use three options to see more information about the processing of macro code and values.

While developing macros, consider using any of three options.

Consider the following program with various macro options invoked one at a time.

Mprint writes the code actually created by the macro syntax to the Log window.

Symbolgen shows the values of macro variables.

Mlogic allows the programmer to trace the flow of the macro execution.
These options are of definite value when determining resolution of multi-pass macro variables. Consider what a headache it would be to figure out the final resolution in the following series.

Without the macro options turned on, the log shows simply the final resolution of each %put statement.

However, more detail is provided with the options turned on.

When not developing macros, efficiency considerations suggest turning off the macro debugging options.

Data Step and Proc SQL Macro Variables

- SYMGET Function
- The Call SYMPUT Routine

Data Step and Proc SQL Macro Variables

This module looks at how the Data Step and Proc SQL can both create macro variables, often out of a data set variable. The Data Step syntax uses the Call SYMPUT routine whereas the Proc SQL uses the into clause. Further, the module looks at the Symget function for bringing a macro variable into the Data Step syntax.

An additional use of the Data Step is to use the Resolve function and the Call Execute routine. The Resolve function is rare; however, the Call Execute function provides a programming option for executing a macro bundle upon a condition being met.

Symget Function

In the beginning, it is very important to realize two things about the Symget function:
- it is used exclusively in the data step
- it works at data step execution time (not compile time).

The purpose of the Symget function is to pass values from a symbol table in the current referencing environment to the data step variable in the program data vector:

<table>
<thead>
<tr>
<th>Symbol Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvar1</td>
</tr>
<tr>
<td>mvar2</td>
</tr>
</tbody>
</table>

Program Data Vector

<table>
<thead>
<tr>
<th>Var</th>
<th>VarB</th>
<th>VarC</th>
<th>VarD</th>
<th>VarE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value2</td>
</tr>
</tbody>
</table>

Use of the Symget function is quite versatile.
The Symget function used in a where option

The Symget function used in a conditional assignment

Consider, in contrast, the following attempt to use the Symget function.

Warning: This program contains syntax errors!

Unlike the Symget function, invoking a macro variable value using &value does not depend on the Data Step nor on execute time.

The Log window shows several problems.

How might the problem in the above syntax be solved?

The three %let statement created sequentially-name macro variables.

The distinction between the two syntax options of the Symget function is in the use of quotes!

Quoted indicates a literal; the string inside the quote is the macro variable name searched for in the scope.

Unquoted indicates a data set variable; the value of the data set variable supplies the name of the macro variable.

Creation of Numeric Variables

The point was made above that the Symget function creates a macro which is a character string. What are the implications of this?

Consider the following syntax which appears to be quite similar. The syntax uses the Input function which is used to convert a character variable to a numeric variable.
The derived variables cost1 to cost4 seem to follow the same syntax. They do not. Explore the differences in the Input statements below.

Why these differences in results? Remember that the Input function uses a character value as the first argument. Both Symget and '20' provide the correct variable type, and thus the correct justification (to the left). These two examples work as predicted.

In contrast, &scale and 20 are numeric values. The Log window states that numeric values have been converted to character. However, the automatic conversion used the Best12. format. As a result, the numbers were converted to strings as "          20". What part of the string contributed to the values of cost1 and cost3? Answer: the first two (blank) spaces. As a result, the values for cost1 and cost3 are missing.

Remember the result from the Symget function is a character string of default length 200. It will be left justified and therefore works as a literal string.

**The Call Symput Routine**

Of all the macro syntax options, the Call Symput routine is the most important and most useful. With Call Symput syntax, we have another way of creating a macro variable and giving it a value, this time from within data step execution.

Remember, the Call Symput routine – like the Symget function – is used exclusively in the Data Step. Further, the macro variable is not available until the data step creating it is complete (i.e., until after the run statement).

The Call Symput routine has two arguments:

- **First argument**: the name of the macro variable
- **Second argument**: the value of the macro variable.

However, the syntax of the Call Symput routine has subtle variations with major implications. In brief, are the arguments quoted or unquoted?

**Example One**

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00002-00023</td>
<td>The derived variables cost1 to cost4 seem to follow the same syntax. They do not. Explore the differences in the Input statements below.</td>
</tr>
</tbody>
</table>

Both arguments are quoted. Argument one creates a macro variable newmacro with the value hello.

**Example Two**

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00003</td>
<td>First argument unquoted and second argument quoted. Argument one references the data set variable 'x' which has the value of 'mvar'. Argument one creates a macro variable mvar with the value greetings.</td>
</tr>
</tbody>
</table>

**Example Three**

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00004</td>
<td>Neither argument is quoted. Arguments one and two reference data set variables 'x' and 'y' respectively. Both data set variables have values. Argument one creates a macro variable mvar with the value Monday.</td>
</tr>
</tbody>
</table>
Example Four

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00003</td>
<td>Only the first argument is quoted. Argument two references a data set variable ‘y’ with value ‘Monday’. Argument one creates a macro variable ‘x’ with the value <em>Monday</em>.</td>
</tr>
</tbody>
</table>

These are the four variations in the Call Symput routine. Now note the timing of creating the macro variable with Call Symput.

<table>
<thead>
<tr>
<th>Line</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>00004</td>
<td>The <code>%put</code> inside the Data Step will not resolve since the macro has not been created.</td>
</tr>
<tr>
<td>00007</td>
<td>The <code>%put</code> after the Data Step will resolve.</td>
</tr>
</tbody>
</table>

Using the Call Symput Routine

The programming power of the Call Symput routine can be appreciated only by seeing its application. Consider the goal of creating macro variables and values for frequencies of distinct values in a data set.

Example One: Create output as shown. It requires both a string and numeric value reflecting the average salary of the data set. Format the string to use in a title statement. Use the numeric value in a Where statement to subset the top half earners into a new data set.

Example Two: Create macro variables to show average age by gender values. Show one average age for males, another for females.

Example Three: The idea is to archive observations more than 30 days old. This example could be adapted to any dynamic file (say one under value is determined by the value in the count variable).
FSEDIT control) where it was important to move old observations into some archive or backup file.

Example Four: Subset a data set so that each distinct value of a variable is written to a new data set bearing the value name.

Frequently Asked Questions

(a) Which symbol table does the macro variable go in?

Most macro variables created by the use of the Call Symput routine are placed in the global table. However, the variable will be placed in the nearest symbol table in the current referencing environment of the data step, providing that symbol table is not empty. If it is empty, it will be placed in the symbol table higher, providing that is not empty and so on.

(b) When is the macro variable available for use?

The most common mistake with the use of the Call Symput routine is to forget that the macro variable is only available after the data step completes execution!

(c) How does the Call Symput format character values?

The default format is $w$, where $w$ is the width of the variable. Hence trailing blanks may also be transferred. Avoid this by the use of the trim function with the second argument:

\[
\text{call symput('mvar1',trim(datavar))};
\]

(d) How does Call SYMPUT format numeric values?

The default format is BEST12, with the number being right justified. You may need to use the left function to get your desired result:

\[
\text{call symput('mvar1',left(datavar))};
\]