STAT 541

Chapter 8 (cont.): Processing Macro Variables at Execution Time

Creating a Macro Variable During DATA Step Execution

Important Reminder: %LET statements are *always* processed by the macroprocessor *before* the DATA step is executed. Ignoring this fact can lead to undesirable results.

Example of Incorrect Coding: %LET statements are processed before the DATA Step is executed

```
data books;
input rank 1. +1 title $21.;
if rank=1 then do;
            %let titletext=Top Bestseller; end;
else do; %let titletext=Other Bestsellers; end;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;

proc print;
where rank=1;
title "&titletext";
run;
```

The SYMPUT Routine

- A CALL routine that can transfer information between an executing DATA step and the macro processor
- Creates a macro variable and assigns *any value available in the DATA step* to that macro variable

The SYMPUT Routine (continued)

CALL SYMPUT(*macro-variable*, *text*);

- Where *macro-variable* is assigned the character value of *text*
- *Macro-variable* and *text* can each be:
  - A literal, enclosed in quotation marks
  - A DATA step variable
  - A DATA step expression

Example of Using SYMPUT with a Literal

- *Myfavorite* is the macro variable.
- Enclose the literal string, "The Hobbit" in quotation marks.

```
data books;
input rank 1. +1 title $21.;
call symput ('myfavorite','The Hobbit');
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
proc print;
title "These are the Bestsellers, but '&myfavorite' is the best!";
run;
```

Example of Using SYMPUT with a DATA Step Variable

CALL SYMPUT('*macro-variable*',*DATA-step-variable*);

- *Topseller* and *rank* are the macro variables.
- *Title* is *a* DATA step variable without quotation marks in the CALL.

Example of Using SYMPUT with a DATA Step Variable

```
data books;
input rank 1. +1 title $21.;
if rank=1 then do; call symput ('topseller',title);
                   call symput ('rank', rank); end;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
proc print;
title "Bestseller List with #&rank-Ranked '&topseller.'";
run;
```

Reminders about Using SYMPUT with a DATA Step Variable

- The values of macro variables are always character strings. Maximum of 32,767 characters can be assigned to the receiving macro variable.
- An automatic numeric-to-character conversion is made using the BEST12. format on any numeric value that is assigned to a macro variable.
- Any leading or trailing blanks that are part of the DATA step expression in the second argument are stored in the macro variable. Use DATA step functions to remove the blanks.

Examples of DATA Step Functions Useful for Removing Blanks or Characters

Remove Characters from Strings

- COMPBL (Removes multiple blanks from a character string)
- COMPRESS (Returns a character string with specified characters removed from the original string)

Remove Blanks from Strings

- LEFT (Left-aligns a character string)
- TRIMN (Removes trailing blanks from character expressions, and returns a string with a length of zero if the expression is missing)
- RIGHT (Right aligns a character expression)
- STRIP (Returns a character string with all leading and trailing blanks removed)
- TRIM (Removes trailing blanks from a character string, and returns one blank if the string is missing)

Example of Using SYMPUT with a DATA Step Variable

The DATA Step expression compress(rank) can also be replaced by put(rank,1.) to achieve the same result.

```
data books;
input rank 1. +1 title $21.;
if rank=1 then do; call symput ('topseller',trim(title));
                   call symput ('rank', compress(rank)); end;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
proc print;
title "Bestseller List with #&rank-Ranked '&topseller.'";
run;
```

The SYMPUTX Routine

CALL SYMPUTX(*macro-variable*,*expression*);

- Where *macro-variable* is assigned the character value of *expression* **AND** automatically removes leading and trailing blanks from both arguments

- *Macro-variable* and *expression* can each be:
  - A literal, enclosed in quotation marks
  - A DATA step variable
  - A Data step expression

Example of Using SYMPUTX with a DATA Step Variable

- Replacing SYMPUT with SYMPUTX in the last example eliminates the need to use the COMPRESS function.

```
data books;
input rank 1. +1 title $21.;
If rank=1 then do; call symputx ('topseller',title);
                   call symputx ('rank',rank); end;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
proc print;
title "Bestseller List with #&rank-Ranked '&topseller.'";
run;
```

Controlling Numeric-to-Character Conversions with the PUT Function

PUT(*source*, *format.*)

- Where *source* is a constant, a variable, or an expression (numeric or character)
- *format.* is any SAS format or user-defined format, which determines:
  - the length of the resulting string
  - whether the string is right- or left-aligned.
- *Source* and *format.* must be the same type (numeric or character)

Examples of the PUT Function in the SYMPUT Routine

- SAS Date variables are numeric variables. Let us suppose that *begin_date* is a SAS Date variable.

```
call symput('date',put(begin_date,mmddyy10.));
```

- Let us suppose that *fee* and *totalpaidup* are both numeric variables.

```
call symput ('due',trim(left(put(fee*(total-paidup),dollar8.))));
```

Creating Multiple Macro Variables During DATA Step Execution

- There might be a need to create many macro variables using one DATA step.
- You can create multiple macro variables in one DATA step using CALL SYMPUT.

The SYMPUT Routine with DATA Step Expressions as Arguments

CALL SYMPUT(*expression1*,*expression2*);

- Where *expression1* evaluates to a character value that is a valid macro variable name. This value changes when another macro variable is created
- *expression2* is the value assigned to the macro variable specified by *expression1*

Example of Using SYMPUT to Create a Macro Variable for Each Record in the DATA Step

- Three global macro variables *rank1*, *rank2*, and *rank3* are defined in the following program.
- Once defined, they can be referenced in other parts of the program.

```
data books;
input rank 1. +1 title $21.;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
data _null_;
set books;
call symput('rank'||put(rank,1.),title);
run;
%put _user_;
```

Referencing macro Variables Indirectly and the Forward Re-Scan Rule

- The macro processor resolves two ampersands (&&) to one ampersand (&), and rescans the reference.
- To re-scan a reference, the macro processor repeatedly scans and resolves tokens from left to right from where multiple ampersands are coded, until no more triggers are resolved.
- Use &&& in front of a macro variable name when its value matches the name of a second macro variable. This indirect reference resolves to the value of the second macro variable.

Referencing Macro Variables Indirectly and the Forward Re-Scan Rule

- Use &&& in front of a macro variable name when its value matches the name of a second macro variable. This indirect reference resolves to the value of the second macro variable.

%call symput(macroref1,macroref2)

&&&macroref1 ➔ [&&]&macroref1➔ [&][&macroref1] ➔ &macroref2

&&macroref1➔ [&&]macroref1 ➔&macroref1 ➔macroref2

Example of Referencing Macro Variables Indirectly

```
data books;
input rank 1. +1 title $21.;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
data _null_;
set books;
call symput('rank'||put(rank,1.),title);

%let titleforrank=rank1;
proc print; var rank;
where rank=input(substr("&titleforrank",5,1),1.);
title "Rank for &&&titleforrank";

%let titleforrank=rank3;
proc print; var rank;
where rank=input(substr("&titleforrank",5,1),1.);
title "Rank for &&&titleforrank";
run;
```

Obtaining Macro Variable Values During DATA Step Execution

- Earlier examples showed how the SYMPUT routine is used to create a macro variable in a DATA step and how macro variable references can be used to assign the value of a macro variable during DATA step execution.
- Obtain or return a macro variable's value during DATA step execution by using the SYMGET function.

The SYMGET Function

SYMGET(*macro-variable*)

Where *macro-variable* can be specified as:

- A macro variable name, enclosed in quotation marks
- A DATA step variable name whose value is the name of a macro variable
- A DATA step character expression whose value is the name of a macro variable

Example of Using the SYMGET Function to Obtain Macro Variable Values During DATA Step Execution

```
data books;
input rank 1. +1 title $21.;
cards;
1 A Tale of Two Cities
2 The Lord of the Rings
3 The Hobbit
;
data _null_;
set books;
call symput('rank'||put(rank,1.),title);

data books2;
set books (keep=rank);
length title $20.;
title=symget('rank'||put(rank,1.));

proc print;
run;
```

Processing Macro Variables during PROC SQL Execution

- Use the INTO clause in a SELECT statement to create or update one or more macro variables
  - Macro variable names are preceded by a colon
  - The INTO clause cannot be used when creating a table or view
- Baseball Data Example

Processing Macro Variables during PROC SQL Execution

```
proc sql;
  select sum(hits)/sum(atbats) format=f4.3 into
    :teamba
  from bbstats;
quit;
%let teamba=&teamba;
proc print data=bastats noobs label;
title "USC Batting Averages";
title2 "Team Batting Average is &teamba';
run;
```

Creating Many Macro Variables with INTO

- The syntax INTO mname1-mnamek can be used to create multiple macro variables in PROC SQL
- It is best to construct code that does not rely on knowing the value of k beforehand, which is often data dependent.

Creating Many Macro Variables with INTO

```
proc sql;
select count(*) into :nrec from tosclaim;
%let nrec=&nrec;
select tos label="Type of Service", nct label="Count",
    totalclaim into :tos1-:tos&nrec,
:nct1-:nct&nrec,
:totalclaim1-:totalclaim&nrec from tosclaim;
%put _user_;
```

Creating a Single Macro Variable from a column with INTO

- An entire column can be saved as a single macro variable in the format of a character-delimited string

```
proc sql;
select distinct tos into :tostype separated by ', '
    from meddb;
%put &tostype;
```