Resolving and Using &&var&i Macro Variables

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

One of the most important aspects of the Macro Language within the SAS® System is its ability to create code dynamically. This allows the developer to write code that will do a series of operations without knowing exactly what those operations will be at execution time.

This presentation will show how double ampersand macro variables are resolved and how compound macro variables can be used to create dynamic code. Examples include using a list in a flat file to create a set of macro variables that control a series of events.

INTRODUCTION

Symbolic or macro variables are stored in memory and are not associated with a data set as are data set variables whose values are stored in the Program Data Vector for a particular data step. This means that macro variables can be used across DATA and PROC steps in a SAS program.

Macro variables are identified in SAS code by preceding the macro variable name with an ampersand. Thus the macro variable DSN will be coded as &DSN.

TECHNICAL TIP: Professional SAS programmers always refer to the ampersand as "amper" when reading code, and only novice programmers pronounce the full word.

The first step in understanding macro variable behavior is to understand how macro variables are resolved. Symbolic variable resolution refers to the process of converting the contents of the variable into symbols or code that can be processed as if it was entered as part of the program.

Because macro variables are resolved before the SAS code is parsed, the macro variable can be used to store snippets of code, variable names, constants, or even pieces of variable names. These macro variables can then be joined together to form new and exciting combinations. When macro variables contain the name of another macro variable or pieces of names of other macro variables the resulting code will often contain two (or more!!) ampersands. The following sections address the issue of making sense of macro variables with multiple ampersands.

MACRO VARIABLE RESOLUTION

A macro variable may be appended to SAS code including variables and data set names. A period following the macro variable is assumed to be a concatenation operator for that macro variable. Periods before a macro variable have no special significance. Double quotes are use when a macro variable is to be resolved inside of a quoted string.

```
%LET SEX=MALE;
DATA &SEX.ONLY;
SET CLASS.&SEX;
WHERE SEX="&SEX";
RUN;
```

This code will resolve to:

```
DATA MALEONLY;
SET CLASS.MALE;
WHERE SEX="MALE";
RUN;
```

It is not uncommon within the SAS Language for some special characters, such as; single quotes, periods and ampersands, to have dual meanings. Dual use is indicated in the code by placing two of the desired symbol. During the resolution of the macro variables the two

symbols are resolved to one. For example a double .. (period) is used when a . (period) is desired in the text.

```
%LET LIBREF=CLASS;
DATA &LIBREF..CLINICS;
SET SOMEDATA;
RUN;
```

This will resolve to:

```
DATA CLASS.CLINICS;
SET SOMEDATA;
RUN;
```

A similar result is achieved when more than one macro variable is joined to form a single result.

```
%LET DSN=CLINICS;
%LET N=5;
%LET DSN5=FRED;
```

The macro variable `&DSN&N` resolves to `CLINICS5`, `&DSN5` resolves to `FRED`. The &&DSN&N combination first resolves to a macro variable (&DSN5) which then resolves to a value (FRED) in a second pass.

DATA DEFINED MACRO VARIABLES

It is not unusual to have a set of operations that must be performed on a series of data sets, variables, or procedures. The control of this process can easily be maintained by using &&var&i operations.

In the following example PROC FSEDIT is to be run on each of a series of data sets. The names of the data sets are stored in a flat file named PAGES.TXT a portion of which looks like:

```
012_015
016_017
018
```

These data set names are placed into macro variables using the CALL SYMPUT routine in a `DATA _NULL_` step.

```
* Load the names of the data sets;
data _null_;
length ii $2;
infile 'pages.txt' missover;

input @1 dsn $8.;
i+1;
ii = left(put(i,2.));
call symput('n',ii);
call symput('dsn'||ii,left(dsn));

run;
```

The character variable ii assigns a sequential number to each data set name. This number then becomes a part of the macro variable name. In this way `&DSN2` is assigned the value of `016_017`. The total number of entries is also retained in `&N`.

USING && MACRO VARIABLES

Typically the macro variables created in applications such as in the above example will be addressed in the form of &&VAR&I where VAR is the root portion of the variable name and the index number is an integer counter.

A macro DO loop is often used to process each of the macro variables. The code below executes a PROC FSEDIT for each data set listed in the text file in the previous example. Notice that the loop counts from 1 to `&N`, the total number of entries. This means that if the list changes all we have to modify is the list itself.

```
%do q = 1 %to &n;
 PROC FSEDIT DATA=dedata.p&&dsn&q  mod
  SCREEN=GLSCN.descn.p&&dsn&q...SCREEN;
 RUN;
%end;
```

Notice that three periods are required in the SCREEN= option because two passes are needed to resolve the `&&DSN&Q`. The first pass compresses the three periods to two and the second pass compress the two periods to one. For `&Q=2` this code resolves to:

```
 PROC FSEDIT DATA=dedata.p016_017 mod
  SCREEN=GLSCN.descn.p016_017.SCREEN;
 RUN;
```

DYNAMICALLY BUILDING SAS CODE

Often code must be written to be flexible in regard to what data sets or even how many data sets are to be used. Macro variables can be used to construct code that is independent of this knowledge prior to execution. In the following example the SET statement is constructed from macro variables.

```
DATA ALL;
SET
%DO I = 1 %TO &N;
 DEDATA.P&&DSN&I
%END;
;
```

Using the data set names from above this code resolves to:

```
DATA ALL;
SET
 DEDATA.P012_015
 DEDATA.P016_017
 DEDATA.P018
;
```

The following example assumes that we want to process each of the data sets in the list using a BY statement. This is fairly easy unless the BY variables differ for each data set. The macro %KEYFLD creates the globalized macro variable &KEYFLD which contains the key variables for the designated data set. The macro argument will be one of the data set names.

```
%global keyfld;
%macro keyfld(pggrp);
 %if &pggrp = 012_015 %then
    %let keyfld = subject dgtyp;
 %else %if &pggrp = 016_017 %then
    %let keyfld = subject sess occ1;
 %else %if &pggrp = 018 %then
    %let keyfld = subject aepga_;
%mend keyfld;
```

The call for this macro will be `%KEYFLD(&&DSN&I)`. The macro variable `&KEYFLD` can then be used or dissected as needed. In the code below the number of variables in `&KEYFLD` are counted and the right most is used in FIRST. and LAST. processing. In the example below each data set has no more than 6 key variables.

```
*determine the list of key vars;
%keyfld(&&dsn&i)
data _null_;
* count the number of keyvars
* save each for later;
str="&keyfld";
do I = 1 to 6;
 key = scan(str,i,' ');
 if key ne ' ' then do;
  ii=left(put(i,1.));
  call symput('key'||ii,
       trim(left(key)));
  call symput('keycnt',ii);
 end;
end;
run;
```

The macro variable `&KEY2` contains the name of the second variable in the list of BY variables and `&KEYCNT` stores the number of BY variables for this data set. These key fields are used in the following DATA step which checks for duplicate observations in the selected data set and assigns a 1 to `&DUPP` when they are found.

```
* Make sure that there are no
* duplicate keys;
%let dupp = 0;
data dupp; set dedata.p&&dsn&i;
by &keyfld;
* determine if this is a dup obs;
if not (first.&&key&keycnt and
        last.&&key&keycnt);
call symput('dupp','1');
run;
```

The above code will locate duplicate observations using the FIRST. and LAST. options even though the programmer had no idea what the variables in the BY statement would be or even what the last variable in the list would be when the code was written. In the above example for `&pggrp = 016_017` the string `FIRST.&&KEY&KEYCNT` resolves to `FIRST.OCC1`. Since there are three variables in the BY statement, `&KEYCNT` is 3, and `&KEY3` is OCC1.

BUILDING FROM A SAS DATA SET

Often the information needed to construct the macro variables is contained in an existing SAS data set. The process of building the macro variables based on a SAS data set is similar to

that used with a flat file.

The following example produces a plot for each region with the region name in the title. A similar result could be achieved using the BY statement and the #BYVAR option, but this method provides more flexibility.

```
%MACRO PLOTIT;
PROC SORT DATA=CLINICS;
BY REGION;
RUN;
DATA _NULL_;
SET CLINICS;
BY REGION;
IF FIRST.REGION THEN DO;
 I+1;
 II=LEFT(PUT(I,2.));
 CALL SYMPUT('REG'||II,REGION);
 CALL SYMPUT('TOTAL',II);
END;
RUN;
%DO I=1 %TO &TOTAL;
 PROC PLOT DATA=CLINICS;
 PLOT HEIGHT * WEIGHT;
 WHERE REGION="&&REG&I";
 TITLE1
  "Height/Weight for REGION &&REG&I";
RUN;
%END;
%MEND PLOTIT;
```

The macro variable `&reg2` will contain the value associated with the second region in the list of regions. The WHERE statement is then used to subset the data in the PROC PLOT step. This code is independent of the number of regions or the values that they take on. The `&&reg&i` can also be used in ANNOTATE labels.

SUMMARY

SAS macros provide powerful and flexible coding opportunities for the generalization of programs. Since macro variables can be used to pass information between program steps or even into SAS/AF® or SAS/FSP®, they provide an excellent vehicle to store lists of data set or variable names. These lists are then accessed by using the double ampersand, &&.

Through the use of && macro variables it is possible to establish a set of symbolic references that can be based on SAS data sets or flat files. These in turn can be used to write SAS code dynamically at the execution of the SAS program.

ACKNOWLEDGMENTS

Several of the examples used in this paper are based on code that was developed for use in a series of clinical trials studies. The author would like to thank Paragon Biomedical, Inc. in Irvine, CA for permission to use this code.

ABOUT THE AUTHOR

Art Carpenter's publications list includes the book *Quick Results with SAS/GRAPH® Software*, two chapters in *Reporting from the Field*, and over two dozen papers and posters presented at SUGI and WUSS. Art has served as a steering committee chairperson of both the Southern California SAS User's Group, SoCalSUG, and the San Diego SAS Users Group, SANDS; a conference cochair of the Western Users of SAS Software regional conference, WUSS; and Section Chair at the SAS User's Group International conference, SUGI.

AUTHOR CONTACT

Art Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199
(619) 945-0613
72212.211@compuserve.com