

The RESOLVE Function - What Is It Good For?

Ian Whitlock, Westat

Abstract

There are four DATA step functions that communicate with the macro facility.

- 1) CALL SYMPUT
- 2) SYMGET
- 3) CALL EXECUTE
- 4) RESOLVE

CALL SYMPUT and CALL EXECUTE send DATA step information to the macro facility. They are both very powerful tools. Numerous SUGI papers have been written on the wonders of CALL SYMPUT. The more powerful CALL EXECUTE is less well known, but again one can find SUGI papers documenting its use.

On the other hand, have you ever seen a paper on SYMGET or RESOLVE?

These corresponding counterparts get information from the macro facility into the DATA step. One often sees SYMGET employed in macro code, but almost never in a place where the code would not be better written without it. Now RESOLVE is a stronger SYMGET, but I do not even remember one program that even uses RESOLVE, other than those provided by the SAS Institute for explanatory purposes.

This paper presents and solves a problem where RESOLVE is crucial to an easy one step solution.

Introduction

Why is it hard to find significant examples using SYMGET? Two conditions usually hold 1) the macro variable is known, and 2) it has the appropriate value during the DATA step compilation. Consequently one does not usually need SYMGET. For example, suppose you want a DATA step variable, DATAVAR, to hold the value of a macro variable, MACVAR. Then one of the following should be appropriate.

```
retain datavar &macvar ;
retain datavar "&macvar" ;
datavar = &macvar ;
datavar = "&macvar" ;
```

When one of the conditions is violated, it is usually the first condition because we don't know (or cannot specify easily) the macro variable. For example, consider initializing a large array with macro

variables. Here all the suggestions above would be awkward, but SYMGET makes it a simple task.

```
array wt (100) _temporary_ ;
do i = 1 to dim ( wt ) ;
    wt ( i ) = symget
        ( 'mac' || left(put(i,3.)) ) ;
end ;
```

A violation of the second condition is discussed in the next section.

RESOLVE is stronger than SYMGET because SYMGET ("MACAR") is equal to RESOLVE ('&MACVAR'). Note the need for single quotes in the case of RESOLVE. With double quotes MACVAR would resolve at compile time and be the argument to RESOLVE instead of resolving at execution time to become the resulting value of RESOLVE as intended. But RESOLVE can do much more because the argument can be any macro expression. For example,

```
var = resolve ( '&mac&i' ) ;
```

or

```
var = resolve
    ( '%scan(&mac,' || put(i,3.) || ')' ) ;
```

are legal expressions. The first could be obtained using SYMGET in

```
i = left(symget("i")) ;
var = symget ( 'mac' || i ) ;
```

But the second example cannot, when the length of the macro variable is longer than 200 bytes. Remember to use single quotes in the RESOLVE arguments in order to hide the macro triggers (%,&) at the DATA step compile time.

Problem: A Subsetting VIEW

To consider an example violating the second condition, suppose you have a DATA step view. The purpose of the view is to subset the input data based on the value of a macro variable, SUBCOND, and then do some calculations. You cannot write

```
data sub / view = sub ;
    set main ;
    where x = "&subcond" ;
    ....
run ;
```

because &SUBCOND would resolve during the compiling of the VIEW, instead of during its execution; hence we have a condition for a significant use of SYMGET. However, SYMGET does not work here because the WHERE statement considers the expression

```
symget ( "subcond" )
```

to be a constant, and for efficiency reasons evaluates it during compile time. Now we need the stronger function, RESOLVE, as in

```
data sub / view = sub ;
  set main ;
  where x =
    resolve('&subcond');
  ....
run ;
```

Problem: Edits given as data values

Suppose you have a list of validation checks given by the character variable COND. For example,

```
COND
u ^= v
x < y or z = 1
....
```

The data values correspond to simple code, but how can one evaluate these expressions to 0 or 1? The answer is to turn these data expressions into macro expressions and then use RESOLVE and %SYSEVALF to evaluate them.

Suppose we replace the values of COND with:

```
&u ^= &v
&x < &y or &z = 1
....
```

Since they are macro expressions they can be evaluated with %sysevalf. In other words, if U, V, X, Y, and Z were macro variables holding the values corresponding to the values of the DATA step variables X, Y, and Z, then the expression

```
resolve ( cond )
```

would be "0" if the expression in COND were false and "1" if it were true.

Putting all this together in a DATA step, produces the following code.

```
data fails ;
  length name $ 8 ;
  set in ;
```

```
array nums (*) X Y Z .... ;
do i = 1 to dim ( nums ) ;
  call vname (nums(i),name);
  call symput (name ,
    put(nums(i),best12.)) ;
end ;
if resolve
  ('%sysevalf('||cond||')')
  = "0" then
  output fails ;

run ;
```

Conclusion

The times when RESOLVE is essential may be relatively rare, but it is crucial for simple solutions at these times. The "edit problem" given above could not have been solved in one step without writing a relatively complex parsing routine.

The author may be contacted by mail at the address:

Ian Whitlock
Westat
1650 Research Boulevard
Rockville, MD 20850-3129

or by e-mail at

whitloi1@westat.com

Bibliography

Riba, S. David (1996). Self-Modifying SAS Programs: A DATA Step Interface. *Proceeding of the SouthEast SAS Users Group Fourth Annual Conference*. SESUG, Atlanta, GA. pp 42 - 49.

Whitlock, H. Ian (1994). CALL EXECUTE versus CALL SYMPUT. *Proceeding of the Seventh Annual NorthEast SAS Users Group Conference*. NESUG, Philadelphia, PA. pp. 254-255

Whitlock, H. Ian (1997) CALL EXECUTE How and Why. *Proceedings of the Twentyfirst SAS Users Group International*. San Diego, CA. pp. 410-413.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.