

Solving with RESOLVE

Minh Duong, David Brodie, and Kevin Davidson
FSD Data Services, Inc.

ABSTRACT

The RESOLVE function provides the capability of resolving a macro variable or a macro reference inside a text string. In many instances, this can help you generalize and simplify your code. We will work through an example of displaying a radio box on an HTML form where the radio box must detect and display what had been previously selected and stored in a SAS data set. Also discussed will be several advantages to using the RESOLVE function instead of SYMGET.

INTRODUCTION

While you may not always love it, the SAS Macro Language is a many splendored thing. Following some careful thought and careful coding, the macro facility enables a programmer to reduce the amount of repetitive coding required. In some cases, the reduction in program code can be quite dramatic. Understanding the basics of the macro language, including macro construction and macro variables is the first step in using the language. As in any other language, once the basics are learned it is time to start learning some of the nuances and subtleties that will further enhance your programming skills. The RESOLVE function is one handy little tool to keep in your back pocket. If you frequently use macro code, you will no doubt have some instances where it will make your job a little easier.

Let's start with a definition and example to show how the RESOLVE function might be of use to you.

First the definition: The RESOLVE function resolves the value of a text expression during data step execution.

Now the example:

Suppose you have a program such as the following:

```
data letter;
  infile cards truncover;
  input text $100.;
  cards;
Hello &recipient,
I hope this message finds you doing well.
Sincerely,
Minh Duong
;
run;

%let recipient=John;
data _null_;
  set letter;
  text2=resolve(text);
  put text2;
run;
```

In the log window, the following would appear:

```
Hello John,  
  
I hope this message finds you doing well.  
...  
Sincerely,  
  
Minh Duong
```

The example shows a simple report. In the LETTER data set, the records form something akin to a mail merge document. Within the letter, there is a field that must be updated, in this case a name value. The name value is stored in a macro variable called 'recipient'. The data _null_ reporting step shows the use of the RESOLVE function. When the "text2=resolve(text)" step is run, the RESOLVE function checks the 'text' field to see if it contains any macro references. For the first record, it finds the macro variable "&recipient" and resolves it, replacing its reference within the text string when writing out to the 'text2' variable.

This example shows the basic functionality of the RESOLVE function. Now let's show some more in depth examples and compare it to some other macro facility options.

VARIOUS USES OF THE RESOLVE FUNCTION

Using SYMPUT and RESOLVE in the same data step

One of the minor obstacles that many a programmer has run into with the use of CALL SYMPUT is that the macro variable being created is not usable until after the data step runs:

```
data _null_;  
  call symput('mymacro',5);  
  y="&mymacro";  
  put y=;  
run;
```

The above program returns the following warning message:

```
WARNING: Apparent symbolic reference MYMACRO3 not resolved.  
y=&mymacro3
```

The message appears because the &mymacro reference attempts to resolve during compilation. Indeed, if the "&mymacro" variable already existed prior to the data step, then the y variable would contain that value rather than the expected value of '5'. The problem is easily circumvented using the RESOLVE function (or the SYMGET function):

```
data _null_;  
  call symput('mymacro',5);  
  y=resolve('&mymacro'); *Note the single quotes;  
  put y=;  
run;
```

The log displays:

```
y=5
```

SYMGET resolves only a macro variable whereas RESOLVE resolves any macro reference

This is one of the handiest uses of the RESOLVE function. You can actually invoke a macro during a data step to return a text string. A couple of examples will demonstrate:

```
%macro holiday_of_the_month(month);
  %if      &month=1 %then Martin Luther King;
  %else %if &month=2 %then Valentines Day;
  %else %if &month=3 %then St. Patricks Day;
  %else %if &month=4 %then Earth Day;
  %else %if &month=5 %then Memorial Day;
  %else %if &month=6 %then Fathers Day;
  %else %if &month=7 %then Independence Day;
  %else %if &month=8 %then National Homeless Animals Day;
  %else %if &month=9 %then Labor Day;
  %else %if &month=10 %then Halloween;
  %else %if &month=11 %then Thanksgiving;
  %else %if &month=12 %then Christmas;
%mend;
data _null_;
  celebration_of_the_month=resolve('%holiday_of_the_month(' ||
    month(today()) || ')');
  put 'Celebrate ' celebration_of_the_month
    'with us on Friday in the lunch room.';
run;
```

Report in the log shows (assuming this was run in August) shows:

Celebrate National Homeless Animals Day with us on Friday in the lunch room.

The example above shows that the 'holiday_of_the_month' macro was executed during the data step. In addition, a parameter was passed to the macro to do some conditional processing.

While the example above shows a useful way to return text to a data step variable, the following example extends the example a bit in order to point out a unique feature.

```
%macro holiday_of_the_month(month);
  data _null_;
    call symput('month',month(today()));
  run;
  %if      &month=1 %then Martin Luther King;
  %else %if &month=2 %then Valentines Day;
  %else %if &month=3 %then St. Patricks Day;
  %else %if &month=4 %then Earth Day;
  %else %if &month=5 %then Memorial Day;
  %else %if &month=6 %then Fathers Day;
  %else %if &month=7 %then Independence Day;
  %else %if &month=8 %then National Homeless Animals Day;
  %else %if &month=9 %then Labor Day;
  %else %if &month=10 %then Halloween;
  %else %if &month=11 %then Thanksgiving;
  %else %if &month=12 %then Christmas;
%mend;
data _null_;
  celebration_of_the_month=resolve('%holiday_of_the_month');
  put 'Celebrate ' celebration_of_the_month
    'with us on Friday in the lunch room.';
run;
```

The above macro returns to the log:

```
Celebrate data _null_; call symput('month',month(today())); run; with us on Friday in the lunch room.
```

It is an important point to note that all non-macro statements (DATA and PROC steps) in macros resolved by the RESOLVE function are returned as text. If the above macro were executed in open code, the 'data _null_' step inside the macro would indeed run as a data step. However, when the macro is invoked using a RESOLVE function, all three statements in the data _null_ step are returned as text. In addition, because the 'month' macro variable does not get assigned, the name of the holiday is not returned. This feature can at times be quite handy for putting complete data steps or procedures into macro variables.

RESOLVE attempts to resolve additional macro references, SYMGET does not

When you have a series of macro variables, frequently it is desired to indirectly reference one within the series. Although this can be confusing, using multiple ampersands allows you to achieve this goal. The following example demonstrates this capability which the RESOLVE function allows. You will note that the SYMGET function does not allow for similar processing.

```
%let november_holiday=Thanksgiving;
%let november_activity=sledding;
%let december_holiday=Christmas;
%let december_activity=caroling;

%let var1=december_;
%let var2=holiday;

data _null_;
  text='&&&var1&var2';
  x=resolve(text);
  put x=;
run;
```

In the log, the following is displayed:

```
x=Christmas
```

If you substitute the following data _null_ step instead of the one shown above, you do not achieve the desired result:

```
data _null_;
  text=symget("&&&var1&var2");
  x=resolve(text);
  put x=;
run;
```

In the log, the following is displayed:

```
NOTE: Invalid argument to function SYMGET at line 272 column 8.
x=
text=  x=  _ERROR_=1  _N_=1
```

If non-existent macro reference, RESOLVE returns unresolved whereas SYMGET is missing

Another difference between RESOLVE and SYMGET processing is the behavior observed when the macro variable being referenced does not exist. When using RESOLVE, the unresolved text string, including the ampersand (or percent symbol in the case of macros) is returned, along with a WARNING message. When using SYMGET, then non-existence of the macro variable leads to a missing value being returned and a NOTE message being displayed in the log. Here is a simple example:

```
data _null_;
  text_resolve=resolve('&september_holiday');
  text_symget=symget('october_holiday');
run;
```

In the log window, the following appears:

```
WARNING: Apparent symbolic reference SEPTEMBER_HOLIDAY not resolved.
NOTE: Invalid argument to function SYMGET at line 296 column 15.
text_resolve=&september_holiday text_symget= _ERROR_=1 _N_=1
```

RESOLVE is very handy for use with SAS/INTRNET to display stored values

Suppose you had an HTML page and one component on the page was a radio button item asking about product quality. The item is just one of many and the desire is to be able to enter data in multiple sessions so that you need to store and redisplay the data upon subsequent viewings of the HTML page. The following is a simple representation of what the single component might look like on the HTML page:

Product Quality
☐ Good
☐ Fair
☐ Bad

A very clean copy of the above HTML source code might look like this:

```
<tr>
  <td>
    Product Quality
  </td>
  <br>
  <td>
    <input type=radio name=quality value=G>Good
  </td>
  <br>
  <td>
    <input type=radio name= quality value=F>Fair
  </td>
  <br>
  <td>
    <input type=radio name= quality value=B>Bad
```

```

</td>
</tr>

```

Assuming that the HTML code is stored in a static file, the goal is to write the word 'checked' within the tag if a value was clicked upon in an earlier session. The trick to accomplishing this is to embed a macro reference in the input tag lines. In the example above, the three input tags are changed to look like this:

```

<input type=radio name=quality value=G %radio_check(name=quality,value=G)>Good
...
<input type=radio name=quality value=F %radio_check(name=quality,value=F)>Fair
...
<input type=radio name=quality value=B %radio_check(name=quality,value=B)>Bad

```

If the file is stored as a static file, then the following program will read in the static file and compare the input tag field name against a like name stored in a SAS data set.

```

*The sole purpose of this macro is to write the word 'checked' when
needed;
%macro radio_check(name=, value=);
  %if &&&name = &value %then %do;
    checked
  %end;
%mend;

```

```

*Simulate data stored permanently in a data set;
data a1;
  quality='F'; *End-user rated the quality as Fair;
run;

```

```

*Create macro variables out of data set variables (macro variable and
data set variables will have same name);
data _null_;
  set a1;
  length varname $32;
  array cvar(*) _char_;
  do i=1 to dim(cvar);
    call vname(cvar(i),varname);
    call symput(varname,cvar(i));
  end;
run;

```

```

*Create the output HTML file;
data _null_;
  infile 'c:\static.html' trunccover;
  input text $100.;
  text=resolve(text);
  file 'c:\dynamic.html';
  put text;
run;

```

Results in the following:

Product Quality

☐ Good

☒ Fair

☐ Bad

An examination of the HTML source code will reveal that the word 'checked' now appears next to the appropriate tag, such as in the following:

```
<input type=radio name=quality value=G >Good
...
<input type=radio name=quality value=F checked >Fair
...
<input type=radio name=quality value=B >Bad
```

CONCLUSION

The RESOLVE function is a very useful utility to use once you have learned the basics of macro processing. There are several unique features that its use offers which can help the programmer achieve the desired results with a simpler syntactical structure and few lines of code.

CONTACT INFORMATION

Minh Duong
FSD Data Services, Inc.
2020 Southwest Freeway, Suite 206
Houston, Texas 77098
713-942-8436
minhd@fsd.nu
www.fsd.nu

Dave Brodie
FSD Data Services, Inc.
2020 Southwest Freeway, Suite 206
Houston, Texas 77098
713-942-8436
daveb@fsd.nu
www.fsd.nu

Kevin Davidson, Ph.D.
FSD Data Services, Inc.
2020 Southwest Freeway, Suite 206
Houston, Texas 77098
713-942-8436
kevind@fsd.nu
www.fsd.nu

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.