MWSUG 2016 - Paper BB-14

WAPTWAP, but remember TMTOWTDI

Jack N Shoemaker, Greensboro, NC

ABSTRACT

Writing a program that writes a program (WAPTWAP) is a technique that allows one to solve computing problems in a flexible and dynamic fashion. Of course, there is more than one way to do it (TMTOWTDI). This presentation will explore three WAPTWAP methods available in the SAS system.

1) %INCLUDE METHOD

The SAS® system allows you to include source code from another system file by using the %include statement. The file referenced by the %include statement is included in the main source code just as if the code were part of the main program. The basic syntax of the %include statement is:

```
%include 'filename.ext';
```

The statement above will include the contents of the file, 'filename.ext', into the SAS® program containing the %include statement. The to-be-included file may be created with a text editor like any other SAS® program, or employing a WAPTWAP technique, using a put statement to write valid SAS® code to the external operating-system file. For example, the following code will create an external file called 'include.sas' with a single PROC PRINT statement.

```
data _null_;
  file 'include.sas';
  put 'proc print;';
  run;
```

The newly-created file, 'include.sas', can then be included using the %include statement.

```
%include 'include.sas';
```

Note that the put statement writes what's between the single quotes to the external operating-system file. In order for the technique to work, the to-be-included file must contain valid SAS® statements. All SAS® statements terminate with a semicolon, so the put statement must write the semicolon to the external file. Hence, there is a semicolon contained within the quoted string. The semicolon outside of the quoted string terminates the put statement itself.

The %include statement may reference a FILEREF as well as a physical operating-system file.

```
filename x ‘include.sas’;

data _null_;
  file x;
  put ‘proc print;’;
  run;

%include ‘include.sas’;
```

Furthermore, as a WAPTWAP technique, the intermediate to-be-included file has no currency past the current session. The TEMP device type instructs SAS® to create a temporary file that exists only for the current session.

```
filename x TEMP;

data _null_;
  file x;
  put ‘proc print;’;
  run;

%include x;
```

This technique has the added virtue of making the code platform independent as SAS® will take care of the operating-system-specific details for the to-be-include file. Another advantage of this technique is that you have the full power of the SAS® data step at your disposal.

There are two disadvantages to this technique. Firstly, it is hard to de-bug. You could encounter a problem due to the included code or due to the way that you create the included code. This leads to the second disadvantage of this approach, namely, it is ill-suited for complex SAS® programming statements. In our trivial example, there is just a single PROC PRINT statement. Even something as simple as adding a title statement adds complexity to the code.

```
data _null_;
  file x;
  put “title1 ‘proc print example’;”;
  put ‘proc print;’;
  run;
```

Since the title statement requires a quoted string, the put statement must use double quotes to quote the single-quoted string referenced by the title statement. You can easily fall into a forest of single and double quotes that become difficult to read and decipher. One way around this problem is to encapsulate the code in a macro and then just ‘put’ the macro calls to the external file. For example,

```
%macro doit;
  title1 ‘proc print example’;
  proc print;
%mend doit;

data _null_;
  file x;
  put ‘%doit’;
  run;
```

With this indirection at least the guts of the SAS® instructions are ‘plain’ SAS® code wrapped in a macro. Of course, if you are going to wrap the SAS® code in a macro, why not implement the entire WAPTWAP technique in macro code itself. This is our second technique.

2) %DO LOOP METHOD

If you only had one PROC PRINT statement to issue, employing a WAPTWAP technique is overkill. Usually you have a list of set of items that you want to process in the same fashion. So, rather than printing just one data set, assume you have a list of data sets that you want to print. If you put this list in a SAS macro symbol, then you can loop over this list to print (process) each item in the list. For example,

```
%macro DoList( LIST= );
%local C DSN; %let C = 1;
%let DSN = %scan( &LIST., &C, ' ' );
%do %while( &DSN ^= );
   proc print data = &DSN.( obs = 1 );
   %let C = %eval( &C + 1 );
   %let DSN = %scan( &LIST., &C, ' ' );
%end;
%mend DoList;
```

This macro loops over a list of data set names referenced in the LIST macro parameter. The list of data set names is parsed into individual data set names and each data set is printed. The code in bold above is the actual generated SAS® code. You can specify the value of LIST by hand; however, WAPTWAP techniques are usually data driven. Assume that you have these data set names in a data set called DRIVER. You can then use PROC SQL to create a space-delimited list of data set names and call the %DoList() macro as follows.

```
proc sql noprint;
  select DSN into :DSN_LIST
    separated by ' '
  from driver
  ;
  quit;
  run;

%DoList( LIST = &DSN_LIST. )
```

The MPRINT and SYMBOLGEN options provide visibility into the generated code for de-bugging. This is an advantage over the first technique. Also, since this is entirely SAS® macro code, you have the full power of the SAS® macro language at your disposal. The disadvantage is that you must wrap the loop in a macro and then call the macro. (Note that a soon-to-be-released version of SAS® will allow for macro %do loops in open code.) More worrisome is that there is a limit on the size of the macro symbol &DSN_LIST. Although as a practical matter this limit is usually not a show stopper, if your driver is quite large, you may run out of room to store all the data set names in a macro symbol. When this occurs, your code will fail with an error similar to this:

```
ERROR: The length of the value of the macro variable DSN_LIST (65540)
exceeds the maximum length (65534). The value has been truncated to 65534
characters.
```

Often the macro symbol will not terminate on a data set boundary, so in addition to not processing the whole list, you will likely encounter a data-set-not-found error. The solution is to employ PROC FCMP to wrap the macro in a data set function that you can call from a normal SAS® data step. This is our third technique.

3) PROC FCMP METHOD

To use PROC FCMP as a macro wrapper, we first modify the %DoIt() macro to make it PROC FCMP-friendly.

```
%macro DoIt( );
%let DSN =
  %sysfunc( dequote( &DSN. ) );

title1 "FCMP RUN_MACRO: Printing first OBS of [&DSN.]";
proc print data = &DSN.( obs = 1 );
  run;
%mend DoIt;
```

With the modified %DoIt() macro in place, we use PROC FCMP to create a data-step function called ‘doit’ which invokes the %DoIt() macro.

```
proc fcmp outlib = work.func.sug;
  function doit( DSN $ );
    rc = run_macro( 'DoIt', DSN );
      return( rc );
  endsub;
  run;
```

With the user-defined doit function in place as an entry in the WORK.FUNC.SUG catalog, we can call the function from a data step as follows using the DRIVER data set as input.

```
options cmplib = work.func;
  run;
data _null_;
  set Driver;
  rc = doit( DSN );
  run;
```

Since we are not generating a gigantic macro symbol to hold the list of data set names, this technique will work with a driver data set of arbitrary size. This freedom comes with several disadvantages. Firstly, the run_macro method only works with positional parameters. This is not a limitation in this example since we only have one parameter to pass along; however, for more complex processing that requires multiple parameters, this limitation could pose some serious headaches. Secondly, the technique requires modification of the original simple %DoIt macro to accommodate the way that the run_macro method operates.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jack N Shoemaker
Greensboro, NC
jack.shoemaker@gmail.com