Paper 105-2010

Pulling Together Existing Code for One-Touch Execution by Using %INCLUDE or Macro References

Mark Menzie, Assurant Health, Milwaukee, WI

Abstract

This paper illustrates the convenience pulling together existing code files into a single stream using %INCLUDE and MACRO references. Steps needed to pull in external code by each method will be illustrated and compared.

Introduction

Often we write a piece of code and discover that it can be used again in other programs. Also, sometimes we may want to consolidate multiple existing programs into a single program for ease of use. This paper explores the use of %INCLUDE and MACRO references to accomplish these goals.

Using Local MACRO references

Before the MACRO can be used it must be defined in the SAS® code. Simple macros can be defined in the program file as follows:

```
%let MACRO1=macrotext;
```

The exact string assigned to MACRO1 in the LET statement will be passed when MACRO1 is called. If quotation marks are used, these will be passed. A more complex macro can consist of n-multiple statements like this:

```
%MACRO MACRO2;

    Macro statement1;

    Macro statement2;

    …..

    Macro statementn;

%MEND MACRO2;
```

Now let's call our macros.

```
% MACRO1 ;

% MACRO2 ;
```

Both macros are called with the same syntax.

%MACRO1; is now replaced in the code stream with *macrotext*. %MACRO2; will be replaced in the executed code by the N statements defined in its MACRO. The value assigned by the macro can be a command, text string, a list, or multiple statements. Some examples are provided in an attachment.

Using Library MACRO references

In addition to including our macros in the current code file we can reference a MACRO library as follows:

```
options MAUTOSOURCE symbolgen sasautos=(sasautos "MACROPATH");
```

If no SASAUTOS library is defined then SASUSER is used by default. When a MACRO is referenced first the MACRO library will search for a matching filename then if none is found the program code will be searched for the MACRO name. If the search for the MACRO does not find a match the MACRO will not return a value and may generate an error in the SASLOG.

Calling a MACRO from the MACRO library uses the same syntax as if the MACRO were locally defined

```
%MACRO1;
```

We don't need to track if the MACRO is local or library defined since each is called the same way.

Using %INCLUDE to consolidate existing programs to run at once

At Assurant we have 22 renewal SAS® Enterprise Guide® code files that we run each day to monitor various reports. Before we redesigned these code files we entered each as a separate project and ran these one at a time. In all it took approximately 30 minutes each day to enter each file, run it, and wait for it to finish.

To ease this burden we re-designed the process to run all the existing programs from a single code file with %INCLUDE. With minimal re-design of the program code we're able to run all these reports at once and save about 20 minutes daily.

Defining the Target Location

Before we can invoke %INCLUDE we need to use the FILENAME statement to define the location of the files to be included. With SAS® Enterprise Guide® on a Windows file system this location is a directory folder. Other platforms can have a different interpretation of the definition invoked with FILENAME.

```
Filename DAILYJOB 'DAILYJOB_LOC';
```

With DAILYJOB defined we can now invoke %INCLUDE

```
%include DAILYJOB(          'JOB1.sas'
                            'JOB2.sas'
                            'JOB3.sas'
                            …
                            'JOBn.sas');
 Run;
```

Each job referenced in %INCLUDE needs to correspond to a file in the folder DAILYJOB_LOC.

Issues to Watch for Using %INCLUDE

When bundling existing code together there are things to watch for. As with all programming changes, keep a backup of code before changes are implemented. Test your change in a controlled environment before releasing it.

With %INCLUDE there are opportunities and issues not encountered with other methods.

The biggest issue that I found in our code was name conflicts. Often the same name would be used in different programs referring to different things. These were corrected by giving each identifier a unique name which spelled out more specifically what it meant. In addition to avoiding conflicts this also made it easier to trace programming issues because error messages would now be more specific to the source of the error.

One issue that we're still struggling with is that running all of the code files at once generates a massive logfile which complicates troubleshooting. One method for addressing this issue is to run only one job at a time like so

```
%include DAILYJOB(          /*      'JOB1.sas'
```

```
                                        'JOB2.sas' */
                                        'JOB3.sas'  /*
                                        …
                                        'JOBn.sas' */ );
      Run;
```

Above, by commenting out all of the child code files except job3 we are able to see if that is where the error occurred. This is possible because all of the jobs are still essentially standalone jobs with little dependence on one another.

One opportunity, in addition to simplifying the task of executing the jobs, is to consolidate overlapping inputs and outputs and avoid duplication. Some of our often used files were being imported several times over in the old jobs, now with a consolidated job these inputs have also been consolidated so that each file is only imported once and then accessed by the child jobs that need it.

Comparing MACRO references and %INCLUDE

The MACRO reference and %INCLUDE act I a similar way. Like many other items in the SAS® toolkit we can do many tasks with either one, or simply copy the code we want to use into the current code file.

The MACRO reference provides a way to repeat often needed code sequences in a shorthand form. In doing so many complex bits of coding can be easily repeated using only a few characters.

The %INCLUDE reference allows us to bundle existing code created in different projects into a single execution run without completely rewriting the code involved.

Resources

This paper is based on use of SAS® Enterprise Guide® on a Windows Personal Computer. Examples may work differently on another computing platform.

Contact Information

Mark Menzie
Assurant Health
501 W Michigan St.
Milwaukee WI 53203
Work: 414-299-6156

Cell: 414-517-8663
mamenzie@hotmail.com