

Paper 2761-2016
Be More Productive!

Tips and Tricks to Improve your SAS® Programming Environment

Marje Fecht, Prowerk Consulting LLC

ABSTRACT

For me, it's all about avoiding manual effort and repetition!

Whether your work involves data exploration, reporting, or analytics, you probably find yourself repeating steps and tasks with each new program, project, or analysis. That repetition adds time to the delivery of results and also contributes to a lack of standardization. This presentation focuses on productivity tips and tricks to help you create a “standard” and efficient environment for your SAS® work so you can focus on the results and not the processes. Included are the following:

- setting up your programming environment (comment blocks, environment cleanup, easy movement between test and production, and modularization)
- sharing easily with your team (format libraries, macro libraries, and common code modules)
- managing files and results (date and time stamps for logs and output, project IDs, and titles and footnotes)

INTRODUCTION

I'm sure you can identify with this... You need to start (and quickly finish) a new program or task or report and you are starting with a BLANK program.

Where to start?? Open the latest program in your “RECENT” list, save as a new name, and start figuring out what applies and what doesn't. Oops – that was someone else's program, and their naming convention isn't the same as yours. Is there a better example program you can use as a starting point??

Now, imagine that you have a **program template** that has all of the typical components that you need.



- Comment block providing documentation of required INPUT, expected OUTPUT, macro variables, dependencies, permissions, QA
- Code Modules such as
 - **Startup** features to
 - cleanup the environment and avoid “carry-over” data, macro variables, and options.
 - password setup for data accesses (databases, etc)
 - log and output control and routing
 - test / production environment control
 - access to format libraries and macro libraries
 - Typical modules (to %INCLUDE or %macroname) for Extraction, Manipulation, Formatting
 - ODS code (to %INCLUDE or %macroname) to control export to Excel, PDF, PPT
 - **Final** Steps such as file permissions, and email alerts.

Wouldn't this make your life easier?

This paper focuses on tips to improve your programming environment by standardizing and modularizing the tasks you typically use. Not only will this get rid of your BLANK program, but it will also enable your co-workers to more easily inherit your processes.

PLAN OF ATTACK

You have a good idea of what you need to do to get started, but, you have two choices

- Start typing  OR Build the *plan of attack*. 

MAINTAIN FOCUS: START YOUR PROGRAM BY BUILDING THE OUTLINE / COMMENTS

Don't just start typing!

Instead, include a standard **template program (comments and code)** that guides you through the details you typically use, such as:

- RequestID - for tracking all of the components of the request (log, program, results, etc)
- Project details, Requester, and date of request
- Programmer along with Contact Information
- Required input (macro variables, SAS data sets, metadata)
- Required output (data, reports, external files, PDF, XLS)
- Formats (labelling, categorization, bucketing) → Format Library
- Macros needed to process data and consolidate results → Macro Library
- Timing (runs first of month, runs after monthly data warehouse load, runs daily after transactional updates, runs every Tuesday and Friday at 8 am)
- Modification Details Section (a template where you include details of changes as they occur)
- Outline of steps you need to complete the request
 - Link to **Startup Modules**
 - What data are required and where will you find data?
 - What data modification, enhancement is needed?
 - What format is required for results?
 - Link to **Final Steps**.

See Figure 1 for an example of a **Template Program**.

```

/*****
RequestID - Purpose of Program - SKELETON CODE TO INCLUDE AS STARTING POINT
Requester: Mary NeedsAnswers
Programmer: Marje Fecht, Prowerk Consulting, www.prowerk.com
Request Date: February 23, 2016
Initial Code: March 1, 2016
Goals: Provide strategy success metrics, including .....
Inputs: REQUIRED
        - macro variables:
            RequestID - identifier of request (use for all files)
            Automate (Y / N) - Use production ID or test ID
            Filepath_Out - location for output: LOG, Results, DATA
            SASData_In - location for SAS Data INput
            SASData_Out - location for SAS Data OUTput
            Metadata_In - location of metadata
            DW_Passwd - password strings for DW access
            Strategy_ID

        OPTIONAL
        - Email addresses for alerts

Outputs: Reports:
        SAS Data:
        Data Files:

Notes: Consider modules for: error handling, email alerts, etc
*****/

/***** MODIFICATION HISTORY *****/
When Who Changes

*****/

/** STEP 1: DEFINE manual INPUTS **/
%let Request_ID = MJFxxxxy;
%let filepath_out = /&Request_ID./; /*location for output: LOG, RESULTS, DATA*/

/** STEP 2: CALL STARTUP PROCESSES **/
%include "insert call to startup program here" ;

/** STEP 3: Extraction specific to Request **/

/** STEP 4: Data Manipulation specific to Request **/
. . . etc . . .

/** STEP FINAL: CALL FINAL PROCESSES **/
%include "insert call to FINAL program here" ;

proc printto ;
run;

```

Figure 1: Example Starter Program

START-UP PROCESSING

What do you typically do at the beginning of every program?

- Clean-up (from any processes that ran prior)
- LOG routing including version control
- Setup appropriate userid and password depending on TEST or PROD environments
- Call permanent format and macro libraries.

Instead of repeating the above steps inside of every new program, modularize your code by putting the above into a “Start-Up” module. A **%INCLUDE** can run this SAS program module.

TIP: Generalize the content of your Start-Up module so that it can adapt to your specific request (file locations, request ID, etc). Macro variables as well as date and time processing are great for generalizing code.

Examples of Start-Up Code

- LOG Routing – with tailored locations and date / time processing

```
** Create a macro variable with the current date and time to use  
    for version control in log and output files **;  
  
%let DateTime = %sysfunc( compress(%sysfunc(today()), yymmddN8.)_  
                        %sysfunc(time(), hhmm6.), ' : ' )  
                );  
  
** route Log to permanent location and version control**;  
proc printto  
  log = "&Filepath_Out.\logs\&stage.\&Strategy_ID._&datetime..log"  
;  
run;  
  
** REMINDER: Turn PRINTTO OFF at end of process ! **;
```

- **Clean-up**

If you are running a sequence of programs that are unrelated, you need to make sure that there are no carryover results that would impact the next program. For example

- If macro variable names overlap, could values from the first program impact how later programs work? If yes – DELETE the macro variables before submitting the next program.
- Could temporary datasets from the first program impact results from later programs? Just to be safe, delete those too!

There are many approaches to the cleanup, and one example will be shared here. For more extensive coverage, see the paper *Creating Easily-Reusable and Extensible Processes: Code that Thinks for Itself* at <http://support.sas.com/resources/papers/proceedings10/004-2010.pdf>.

The following macro

- deletes all temporary SAS files(catalogs, data, etc) located in the WORK library
- locates and deletes all macro variables, except macro variables provided in an exclusion list.

```
%macro cleanUp;
*****;
*** clean up work datasets and all pre-existing macro variables;
*****;

*** clean up the work directory ***;
proc datasets lib=work nolist nowarn nodetails kill;
quit;

*** dynamically build list of all macro variables to clean up ***;
data _null_;
  length cmd $200;
  set sashelp.vmacro; /*Use the SAS help macro dictionary table*/
  /*Select all GLOBAL Macro variables with an offset of zero*/
  where
    scope = 'GLOBAL'
    and offset = 0
    and
      /*macro variables NOT included in our cleanup (system vars)*/
      /*You may want to add your "control variables" to exclusions*/
      (      name ne: 'SYSDB'
        and name ne: 'SYSODS'
        and name ne: 'SYS_'
      )
  ;
  /*Cycle through the list of macro variables
  and delete the ones which have been included*/
  cmd = '%nrstr(%symdel ' || trim(name) || ' / nowarn );';
  call execute(cmd);
run;

%mend cleanUp;

*** Call Macro ***;

%cleanUp;
```

COMMON CODE MODULES

Specialized code modules make your programming life easier. In addition to the start-up processing discussed above, what are some other examples of widely used code that you can %INCLUDE as needed?

- Setting file permissions when your job completes. This lets you make sure that others in your team can access the files you have created!
 - Unix Example:

```
x "chmod 777
    "&Filepath_Out.\logs\&stage.\&Strategy_ID._&datetime..log";
```

NOTE: Beware of CASE in your path and file name - unix cares

- Emailing completion updates
 - Are you in the habit of continually checking to see if your jobs have completed?
 - Let SAS email you! As a final step in your program, send an email to yourself (and others who need to know) advising that the job has completed, and also providing a recap of job statistics and any issues.

MACRO LIBRARIES

You and your team probably have many macros that you copy into your programs. Have you considered storing them in a common location, so that there is a single version of the macros that all can use?

There are many approaches to sharing macros including

- Macro libraries with the AUTOCALL facility
 - This is automatically available for the SAS AUTOCALL library, where SAS provides a number of very handy macros.
 - Using the AUTOCALL option, you can add your own macro libraries to the list.
- Compiled Stored Macros

For a good discussion of the approaches, and best practices, see *Carpenter's Complete Guide to the SAS Macro Language, Second Edition* (http://www.sas.com/store/prodBK_59224_en.html).

CONCLUSION

As the requests for results increase and the demand on our time also increases, we have to find ways to streamline our processes, and reduce start-up time. This paper has shared a variety of suggestions to improve your programming environment by standardizing and modularizing the tasks you typically use.

ACKNOWLEDGMENTS

While inherited code can cause headaches, I always enjoy inheriting code so I can learn new best practices, practice improvements, assess ROI, and share enhancements. I appreciate all of the examples over the past 35 years including the good, the bad, and the really ugly.

RECOMMENDED READING

Related papers of interest:

- http://www.sascommunity.org/wiki/Presentations:Marje_Papers_and_Presentations
 - *THINK Before You Type... Best Practices Learned the Hard Way*
 - *Creating Easily-Reusable and Extensible Processes: Code that Thinks for Itself*
 - Video of this presentation from SAS Global Forum 2016
<http://sasgf16.v.sas.com/detail/videos/breakout-sessions/video/4852598989001/2761---be-more-productive-tips-and-tricks-to-improve-your-sas-programming-environment>

CONTACT INFORMATION

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

Marje Fecht, Prowerk Consulting

Marje.Fecht@prowerk.com

http://www.sascommunity.org/wiki/Presentations:Marje_Papers_and_Presentations

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.

Other brand and product names are trademarks of their respective companies.