Paper 34-27

A Modular Approach to Portable Programming

Michael A. Litzsinger, Quintiles Inc.
Michael A. Riddle, Quintiles Inc.

This paper presents SAS® programming techniques that allow operating system and project assignments to be made centrally, but still allow portability to another system. Modules generally include anything that can be best defined in a single location, such as LIBNAMEs, TITLEs, FOOTNOTEs, or system options.

*This paper also details the use of **MFILE,** in conjunction with **MPRINT,** to direct executed program statements to a stand-alone output program. MFILE is a relatively unknown system option that was known as RESERVEDB1 in SAS 6.12.*

The result is a blended method to port code using the output directing capabilities of the MPRINT system option. The resulting SAS program can be stripped of all traces of modularity and underlying macros, and can be executed as a stand-alone program. The combination of these two techniques retains the flexibility of modularity for development and also results in truly portable SAS programs.

Introduction:

Developing code on one platform and then running it on another is not automatically intuitive. The characteristics of each platform may be different. Development platforms tend to be changing, team-oriented environments. Production environments tend to be one-program-per-purpose, static environments. In cases where programs are developed for a client, there are issues on delivering programs "as is". The developer doesn't want to lose control of proprietary development tools. The client doesn't want to learn each developer's methods: they just want to run task-specific programs.

Modular programs (for development)
- Best for medium and large solutions
- Breaks down complex problems
- Global modifications are easy
- Flexible structure encourages innovation
- Eliminates/reduces re-inventing the wheel
- Environment conducive to sharing work

Stand-alone programs (for production)
- Best for small or static solutions
- All detail is present in each program so it can be straightforward to figure out
- Large problems can be overwhelming
- Widespread changes are repetitive
- Code is copied as many times as needed
- Innovation is difficult to implement

Classifying program modules:

To show what is meant by "modularity", here is a program organized into 5 distinct modules**:**

```
*(1) System options;
%include "c:\pgmlib\options.sas";

*(2) Define project libraries/catalogs;
%include "c:\pgmlib\libnames.sas";

*(3) Define project/program specifics;
%include "c:\pgmlib\titles.sas";
%include "c:\pgmlib\tnum.sas";
%include "c:\pgmlib\foots.sas";

*(4) Define macro library;
%include "c:\pgmlib\age.sas";
%include "c:\pgmlib\freq.sas";
%include "c:\pgmlib\npct.sas";
%include "c:\pgmlib\maketbl.sas";

*(5) Main program;
data agedata;
  set datalib.demo;
  age=%age(birthdt,visitdt);
run;
%freq(indsn=agedata,colvar=pop,total=Y,
        npct=%npct("000 (000%)"),out=freqout1);
%freq(indsn=agedata,depvar=age,
        colvar=pop,rowvar=sex,total=Y,
        npct=%npct("000 (000%)"),out=freqout2);
data tbldata;
  set freqout1
      freqout2;
run;
%maketbl(indsn=tbldata,
    method=ProcReport,
    style=Standard3,source=Y)
```

System Options
Placing system options in their own programs facilitates making project-wide system changes. For example, during SAS Y2K remediation each project had to incorporate the appropriate cutoff year. With modules, it was only necessary to set the YEARCUTOFF option in a single place.

Library Names and Formats
A single location for librefs and formats is a natural solution for portability. Projects can easily share programs on different platforms or directory structures. Referencing unique librefs makes it unnecessary to update individual programs.

Titles, Tables Numbers, and Footnotes
Placing project specific details in one location facilitates making quick changes. This technique makes it easy to renumber unplanned tables and to reuse code across similar projects. Differences can be clearly specified, aiding understanding as well.

Macros
The primary use of macro language at all programming levels is the ability to use code repeatedly. Modularizing macros makes them more reusable with availability to all programs within a project or even an organization.

Main Program Body
This is the primary module. It contains code that is unique to the purpose of the program. It usually brings in data and manipulates and/or presents it. The main program brings together all of the sub-modules by calling or passing parameters to them.

A portable program standard:

The techniques presented here follow these two rules:

- Define all "settings" upfront (these are bolded below). Generally, these settings are the first 2 (or 3) modules covered above. It is ideal to keep these settings modular to facilitate porting to a new system.
- Then use MFILE to direct the remaining or "core" program code into a complete portable program. The core program contains just pertinent statements and is absent of "settings" and macros in any form.

For example:

```
%include "S:\proj123\options.sas";
%include "S:\proj123\libnames.sas";

title1 "Table 10.1.1 (Draft)";
title2 "Demographics Summary";
title3 "Intent-to-Treat Population";
footnote1 "Source: DEMO_ITT.SAS";
data agedata;
  set datalib.demo;
  age= (year(visitdt)-year(birthdt))-
          (month(visitdt)<=month(birthdt))+
           (month(visitdt)=month(birthdt) and
          day(visitdt)>=day (birthdt)));
run;
proc freq data=agedata;
tables pop*age/out=freqout;
run;
data tbldata;
  if _n_=1 then set pct;
  set freqout;
  array n n1-n3;

 [… et cetera …]
```

The GOAL – a Portable Program Standard
The goal is to deliver flexible programs, each with a clearly defined purpose. This process makes it easy for a third party to run and quickly understand these same programs on another system or platform.

Generating Code Using MFILE:

In SAS 6.12, "MFILE" debuted as system option RESERVEDB1. It became MFILE in SAS 7.

The MPRINT option is traditionally used as a technique in debugging or to display all executed statements to the log. MPRINT sends macro generated code to the log with the prefix MPRINT beginning each line:

```
MPRINT(PGMCODE):  libname datalib "c:\datalib";
MPRINT(PGMCODE):  data newdemo;
MPRINT(PGMCODE):  set datalib.demo;
MPRINT(PGMCODE):  dsvar=12;
MPRINT(PGMCODE):  run;
```

This code has all macro variables and references resolved and can be thought of as source or "compiled" code reduced to its simplest form. A great feature of MPRINT is that this code can be directed to an external file.

SAS program code is redirected to an output destination when all 3 conditions are TRUE:

- FILENAME MPRINT *path* is defined
- MPRINT and MFILE are in effect
- Code is executed through a macro

The required parts are bolded below. Using this technique, the macro facility saves every executed statement to a file for you.

Running this code:

```
filename mprint "c:\pgmlib\newdemo.pgm";
options mprint mfile;

%macro pgmcode;
   libname datalib "c:\datalib";
   %let var1=12;
   data newdemo;
      set datalib.demo;
      dsvar=&var1;
   run;
%mend pgmcode;
%pgmcode;
```

creates this executable code (newdemo.pgm):

```
libname datalib "c:\datalib";
data newdemo;
   set datalib.demo;
   dsvar=12;
run;
```

By now you may have realized an obstacle: SAS will automatically resolve *all* macro elements in the code it redirects to your saved file. You can't have it selectively pass code that contains desired macro elements, such as %include. Because our goal here is to create stand-alone code containing *some* modular elements (the ability to change system options and librefs), we have to be creative in how and when MFILE is used. Generally, we want to use it to create a stand-alone file of the proprietary elements, and then tweak this file so it can run on a new system. There are several ways to do this, and most methods have a "burden" of knowing the new operating system and location.

The best way to end the redirection of statements is by ending the macro. Other methods write an unwanted remnant to the redirected program:

- OPTIONS *NO*MFILE; or OPTIONS *NO*MPRINT; will become the last statement written to the output program.
- FILENAME MPRINT *path*; becomes the last statement written to the first output program. Then the second output program picks up where the previous one left off.

An important decision that must be made upfront is which type of comment text should be used in your source programs. Not all types of comments are passed to the stand-alone program:

Comment type	Result
*Comment;	Passed through as is: *Comment;
*Comment1 *Comment2;	Text will flow to long lines: *Comment1 *Comment2;
/*Comment*/	Ignored
%*Comment;	Ignored

Thus, only "*" comments are passed unchanged through to the MPRINT file. To avoid long comments on a single line, always end each line with a semicolon. Documentation blocks need to use "*;" comments to neatly pass them through to the stand-alone program. Comments in macros should not use "*" comments as these are not desired in the stand-alone program (proprietary elements should be kept transparent).

Outlined in the following table are solutions that make stand-alone programs created by MFILE more portable. The simplest are the "Interactive Program", "Text Replacement", and "Portable Media" methods. The most complicated is the "Append" method. The method of choice depends on the knowledge of the target operating system, the degree of desired portability, and the client's ability to alter and run SAS programs.

MFILE Methods to Create Portable Stand-alone Programs:

Method	Action	Fully Portable?
Interactive Program	Set librefs once per session	No – requires interactive mode
Comprehensive Program	Create a master program in a directory	No – runs all programs in directory
Text replacement	Replace token text in each program (a text replacement utility makes this easier)	Yes
Append	Append settings with an additional program	Yes
Portable Media	Define and leave on portable media (Zip disk or diskette)	Yes, restricted – must run directly on re-writeable media (this rules out CDs)

Prologue to Methods:

These methods require the program created by MFILE to be stripped of all operating system specific code such as librefs. Each method then has a different way to attach settings necessary for running the code on the new system. Note that the librefs reside outside of the macro that redirects code to the stand-alone program, so they are present for running the code on the original system but are not included in the output stand-alone program.

Running this code (newdemo.sas):

```
*Code not directed to the output program;
%let pgmname=newdemo;
%include "c:\pgmlib\libnames.sas";

*set up for mfile;
options mprint mfile ls=230 ps=80;
filename mprint "c:\pgmlib\&pgmname..pgm";
```

```
*Code directed to the output program;
%macro pgmcode;
%let var1=12;
 data newdemo;
    set datalib.demo;
    dsvar=&var1;
 run;
%mend pgmcode;
%pgmcode;
```

creates this stand-alone code (newdemo.pgm):

```
data newdemo;
   set datalib.demo;
   dsvar=12;
run;
```

Interactive Program Method:

For this method, librefs define temporary work files. Individual programs merely call work files and are run subsequently. Librefs must define work files at the start of every session. The Interactive Program method isn't really a viable solution in a validated program environment, so an example is not provided, but it could be an option in some situations.

Comprehensive Program Method:

For this method, a comprehensive program is built in addition to each individual program provided. The logical arrangement is to link similar tasks or group all the programs from a subdirectory. So if you had 20 summary table programs in a subdirectory, you would create another program that called the 20 programs sequentially. If the operation system and location were known in advance, the comprehensive program could be built upfront. If not, the client user would have to fill in information for file locations (librefs) and the location of the included program library.

This method is not a true stand-alone because all programs are launched from one program. But the simplicity of this approach means that the librefs only have to be defined once at the top of the comprehensive program. This approach may be clear enough to the client that they are willing to fill in the location information. For individual program execution, just comment out unwanted macro calls. For the same result, submit a highlighted selection interactively.

To run this code below, either the program and data locations need to be defined upfront or the client must fill in the information:

```
%let srcpath=c:\pgmlib\;
%let datpath=c:\datalib\;

options nocenter ls=150 ps=55 yearcutoff=1910;

libname datalib "&datpath";

%macro runall(&pgmname=);
  proc printto file="&srcpath.&pgmname..tbl";
  run;
  %include "&srcpath.&pgname..pgm"
%mend runall;

%runall(newdemo);
%runall(neweff);
[… et cetera …]
```

Text Replacement Method:
The most straightforward method for creating portable stand-alone programs is to place a token string at the beginning of each source program. This token can then be replaced or swapped out prior to running the program on the new system once it is determined where datasets and included programs will reside. However, for this token text to be benignly passed through, it must be defined as a "*comment;". Remember, you cannot pass "%include …" through because it is resolved by the MPRINT facility. Other types of comments are not passed through at all.

Running this code:

```
filename mprint "c:\pgmlib\newdemo.pgm";
options mprint mfile;

%macro pgmcode;
   **** new libnames go here ****;
   %let var1=12;

   data newdemo;
      set saslib.demo;
      dsvar=&var1;
   run;
%mend pgmcode;
%pgmcode;
```

creates this code (newdemo.pgm):

```
**** new libnames go here ****;
data newdemo;
   set datalib.demo;
   dsvar=12;
run;
```

A swap or replacement in each program will need to be made.

Replace the text:

```
**** new libnames go here ****;
```

with the text:

```
%include c:\pgmlib\libnames.sas";
```

and define librefs centrally in libnames.sas.

This process can be automated using operating system specific utilities, or each program can be modified individually. For example, a DOS com file or Visual Basic macro can make such changes automatically. On an OpenVMS system, a SWAP com file can easily make large-scale replacements from one command.

Thus, if the target operating system and file locations can be determined in advance, the stand-alone programs can be readily created prior to shipment. Alternately, if the client is able to make the changes, they could determine the locations.

Append Method:
In the append method, a library name module is appended to each stand-alone program generated by MFILE. The following code (append.sas) attaches libnames.sas to newdemo.pgm to create the new program newdemo.sas.

The append.sas program makes use of put, input, and file statements to build the portable program that contains both the libnames.sas component and the stand-alone program created by MFILE. It is only necessary to update the macro variables DATPATH and PGMPATH in append.sas. The libnames.sas program uses these macro variables to set the librefs.

Run this program (append.sas):

```
%let pgmpath=c:\pgmlib;**target programs;
%let datpath=c:\datalib; **data directory;

%macro append(pgm=,srcpath=);
filename pgmout "&srcpath.&pgm..sas";
data _null_;
    file pgmout;
    put @1 "%" "let pgmname=&pgm;";
    put @1 "%" "let srcpath=&srcpath;";
    put @1 "%" "let datpath=&datpath;";
run;
data _null_;
    infile "&srcpath.libnames.sas" pad;
    input @1 line $256.;
    file pgmout mod;
    put line;
run;
data _null_;
    infile "&srcpath.&pgm..pgm" pad;
    input @1 line $256.;
    file pgmout mod;
    put line;
run;
%mend append;
%append(pgm=newdemo,srcpath=&pgmpath);
```

along with this code (c:\pgmlib\libnames.sas):

```
libname datalib "&datpath";
libname library "&datpath";
```

to create this executable code (newdemo.sas):

```
%let pgmname=newdemo;
%let srcpath=c:\datalib\;
%let datpath=c:\pgmlib\;
libname datalib "&datpath";
libname library "&datpath";
data newdemo;
   set datalib.demo;
   dsvar=12;
run;
```

Note that a single append.sas program can be constructed to process multiple programs. The append method is probably the best all-around solution. It can be set up to prompt for dataset and program locations in an intuitive manner.

Portable Media Method:

Packaging all programs and data on re-writeable media (zip drive, diskette) allows programs to be run in place. The programming environment is dependent on the media. There must be ample space available to write back to the disk when required (creating files, table output, SAS system writes, etc.).

The resulting code is "ready to run" and would need no manipulation at run time. The programmer could develop the code directly onto the delivery media. Alternately, it can be created in a testing area and then the text replacement or append method can be used to prepare the code for porting to the new media. This method is really only an option for smaller projects unless an external hard drive is delivered, or higher-density portable read/write storage becomes available.

Conclusion:

The optimal solution combines the flexibility of modular programming with the clarity of stand-alone programming. This solution must provide comprehensive code that still allows proprietary work to be preserved. Providing system options and librefs in a modular format allows true portability to another platform or system. Other components are then delivered as "compiled" non-proprietary code, void of all macros, so that the entire set of programs serves as documentation. This permits easy replication of work. This approach is the most intuitive and is fairly straightforward.

An alternative is to also provide the project specific details (table numbers, titles, and footnotes) to the client in a modular format. This would allow renumbering and other cosmetic modifications to be made by the client, but this depends on the purpose of the ported code and the roles of those involved. This risks overwhelming the client. The system option and libref components also can be largely pre-built should that information be determined as a project is being set up. Changing the degree of modularity provided in the portable programs "after the fact" is much more time-consuming.

Version Note:

"MFILE" first appeared as a system option named RESERVEDB1 in SAS version 6.12. Beginning with SAS version 7 this system option was renamed MFILE. The functionality remained the same. Companies running both 6.12 as well as a more recent version will need to remember this and to change the system option name when moving programs across versions.

Contact Information:

Michael A. Litzsinger
Michael A. Riddle
Quintiles, Inc.
Statistical Programming
P.O. Box 13979
Research Triangle Park, NC 27709-3979
(919) 998-2888
mike.litzsinger@quintiles.com
michael.riddle@quintiles.com