PharmaSUG2011 – Paper SAS-AD03

SAS Drug Development Program Portability

Ben Bocchicchio, SAS Institute, Cary NC, US
Nancy Cole, SAS Institute, Cary NC, US

ABSTRACT

A Roadmap showing how SAS code developed outside of the SAS® Drug Development system can be used with the Process Editor (within SAS Drug Development) and how code developed within SAS Drug Development can be utilized outside of the application.

In an era where going green and recycling are part of everyday life, can SAS code also adhere to these principles in terms of recycling and reuse? If so, how can you take advantage of existing SAS code (recycle it) and make it run in SAS Drug Development's Process Editor? How can this be accomplished with a minimum amount of code changes? In an equally import design decision, if the SAS code generated from the SAS Drug Development system is a required deliverable, how can the code be designed to be able to run outside of SAS Drug Development?

These are some of the many questions that face programmers using SAS Drug Development, an integrated 21CFR Part 11 compliant system for managing, analyzing, reporting and reviewing clinical research information. Fortunately, the design considerations used to make existing SAS code run within SAS Drug Development's' Process Editor employs the same strategy used to allow SAS code to run outside the SAS Drug Development system. This paper explores techniques such as parameter substitution / relative paths that can be used to facilitate code reuse.

INTRODUCTION

This paper describes two different strategies. The first involves developing code outside of SAS Drug Development's environment and then integrating that code into SAS Drug Development with little-to-no code changes. The second strategy revolves around taking code developed within the Process Editor and exporting the code to a client or regulatory body, where the SAS code runs independently of SAS Drug Development, yet produces the same results.

Both of these related strategies employ a programming technique that has been used in the industry for some time. It centers on the ability to 'include' a file that has all of the library references to data, SAS macros, SAS catalogs, etc., that a program needs to run. The file location information is separated from the actual data manipulation steps defined in the code. Use of standard library references and data set names/variables, allows the code to be highly reusable. Therefore, when there is standardization, only minor changes to the 'include' file are required to run the same program for a different set of data. Reusing code saves a great deal of development and validation time.

SAS Drug Development has this same ability to include an 'include' file into a program making it highly reusable. SAS Drug Development takes this reusability advantage one step further through the use of relative paths. Instead of hard-coding a path to pick up the include file (for example, `%include "c:\temp\setup.sas"`), the Process Editor allows the include file to be picked up relative to the location of the calling program. If the 'include' program is in the same directory as the calling program, then the reference to the file is notated as a period (.) in the same relative area. If the include file is in a folder one level removed named 'driver', then the notation in the calling program would state (`./driver`). This means that if you follow a standard naming hierarchy for compounds/protocols, the programs can just be copied to the new protocol location, and just the include file needs to be altered.

LEVERAGING LEGACY SAS CODE TO FUNCTION IN THE PROCESS EDITOR

The goal of SAS Drug Development is to provide users with a 'System' where a SAS program can be executed within a compliant, controlled environment. Within the context of SAS Drug Development, control means only designated user(s) can run the code or have access to the results. Therefore, the ability to leverage preexisting SAS code streamlines the transition in SAS Drug Development.

CODE ANALYSIS

Code Analysis includes the processes of identifying all the points within the SAS code that touch data (LIBNAME statements), use external code (SAS macros), provide format look-ups (format catalogs), or reference any other items that are external to the SAS program. These other items could include input files such as a title database.

As stated in the introduction, your company may have a programming practice to place all of the 'externally needed' file references into one separate file. In this paper, this is referred to as a setup file. This setup file is then included in a standard program and drives the location of the data to use, the macro library paths the program needs to function correctly. If your

program is not initially set up this way, then these bits of information need to be extracted into a separate file to enable the conversion to function correctly. At this point, it may be easier to upload the program as-is into the Process Editor and make the necessary parameter substitution within its interface. This strategy, however, will not facilitate a high degree of reuse.

LOCAL PC SAS CODE MODIFICATION

If your local SAS PC code was designed with a `%include` file, you can take advantage of the following macro. Just include the macro below at the top of your program and make any necessary substitutions to the line of code that contains the pointer to your `%include` file:

```
  %include "s:\project_a\protocol_b\setup.sas";
```

Figure 1: %include Macro

This code points to the `%include` file to your local (network) location. After this change has been made, your code should run the same way it did before the modification.

```
%macro setup;
/*  the macro variable _SDDUSR_ is a SAS Drug Development automatic macro variable */
%if %symexist(_SDDUSR_)  %then %do;
  %put "inSDD";
 %include &setup;
  %end;
%else %do;
   %put "non inSDD";
   %include "s:\project_a\protocol_b\setup.sas";
   %end;
%mend;

%setup;
```

Figure 2: Code Containing %include Macro

However, this step does not complete all the necessary changes to enable the local SAS code to run within the Process Editor. In order for this code to run in SAS Drug Development, you must provide additional metadata. This involves inserting a previously defined piece of XML at the bottom of the program. The XML metadata comes in the form of a SAS comment statement. Because this XML metadata is only a comment, it does not interfere with running the SAS code in the local PC environment.

REVERSE ENGINEERING

Creating the XML metadata to be inserted into the bottom of the local code requires the most planning. You must design the XML metadata to work in the specific SAS Drug Development instance where the PC code will be promoted. Dependences exist on hierarchy design setup in SAS Drug Development, the location of macros, and all the other touch points outlined in the Code Analysis phase. You must account for all of these touch points either within the setup (include file) or within the XML metadata.

For example, when running in the Process Editor, the XML can be setup to pick up the setup file from the same folder the program is uploaded into (by relative path), or the LOG file can be written to a default, `./LOG` subdirectory. However, this `./LOG` subdirectory (case sensitive) must exist in the hierarchy for the code to run correctly.

The XML metadata is constructed through reverse engineering. A sample program is built into SAS Drug Development for the sole purpose to build the metadata. This metadata is then extracted and made available for the local PC SAS code to use.

BUILDING THE %INCLUDE FILE IN THE PROCESS EDITOR

The first step in reverse engineering this XML metadata involves defining the `%include` file that will be used within SAS Drug Development. This is accomplished by loading the existing local PC version of the `%include` SAS code into the system. Open this code into the Process Editor and follow the standard convention for converting local references to parameterized Process Editor Macro variables. Figure 3 (below) shows the result of converting a local PC `%include` file into a SAS Drug Development Process Editor program.

As shown in Figure 3, various levels of SAS macro libraries can be searched, the location of the SAS format catalog can be defined, and the SAS filename and library reference can be set for both data input and results output. All these references are built by first defining a BASE PATH. A *BASE PATH* is the anchor to which a relative path can be defined.

The Default column in the Parameter panel shows the BASE PATH as a fully qualified path. All the rest of the filename,

LIBNAME references use this base path to determine their destinations.

For subsequent clinical trials, if the SAS Drug Development hierarchy is built from a standard, any programs using this `%include` file can be copied to the new clinical trial hierarchy, and the only code change that is necessary is changing the BASE PATH in the `%include` file.

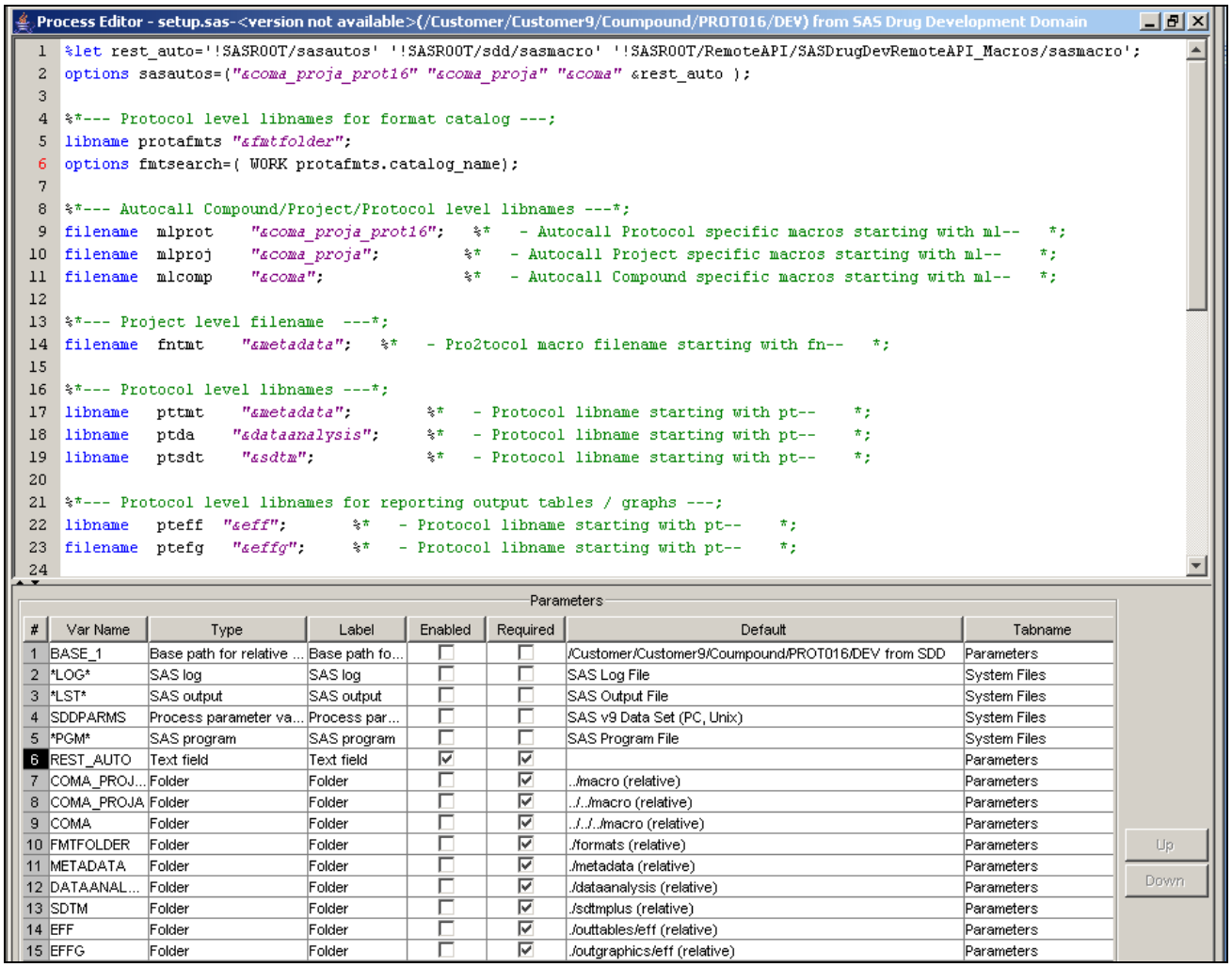

Figure 3: Converting a Local PC %include File into a SAS Drug Development Process Editor Program

CAPTURING THE %INCLUDE FILE

The next step uses the newly created `%include` file to construct the sample 'calling program'. This 'calling program' executes the standard SAS data processing to build output data sets (for example, analysis and SDTM data sets) or output Figures, Tables, and Listings. Note: This example does not include any SAS data transformation or reporting code - just the one statement that includes the setup.sas file.

Figure 4 displays the 'calling program' with just the `%include` statement. **Note**: Once the `&setup` parameter is defined to the setup.sas program (line #5 in the Parameters section of the UI), all of the metadata from the setup.sas program is also included within this sample.sas program.

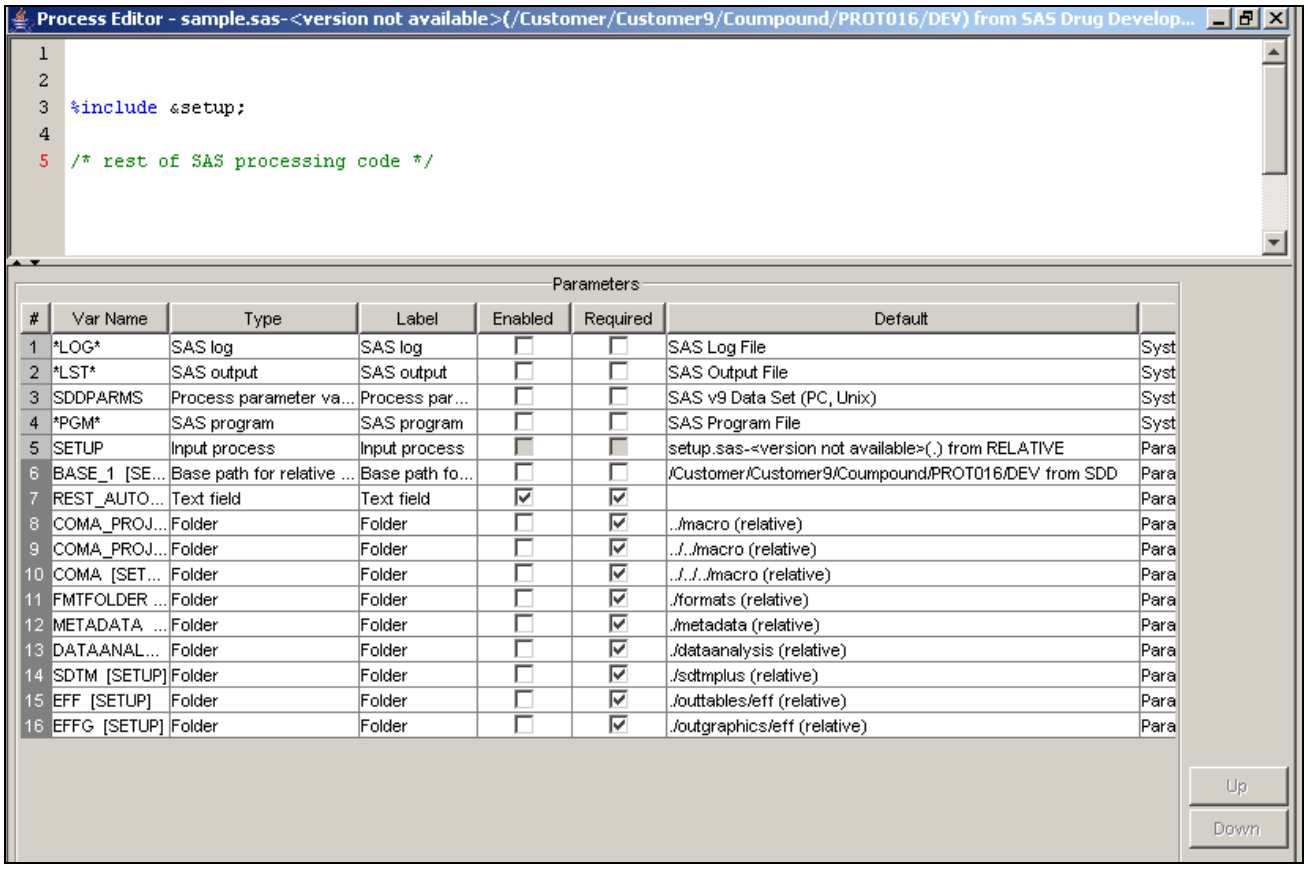

Figure 4: Calling Program with the %include File

Depending on the design of the calling program, you must designate the output location of the LOG and LST files. Two methods exist to setup paths for this output. One method is to have a SAS Library defined with a PROC PRINTTO statement to allow the LOG and LST files to be directed to a desired location.

```
proc printto log="&path/&reportNum.log" new;
run;

proc printto print="&path/&reportNum.lst" new;
run;
```

Figure 5: Use PROC PRINTTO to Setup Paths for SAS, LOG, and LST Files

Note: In the above example, `&path` is the macro path to the output location for the LOG and LST files, and `&reportNum` is the name of the calling program.

A second method defines the metadata for the LOG and LST file to set the desired output location as part of the Process Editor's metadata. This is done within the process Parameter fields for the LOG and LST parameters. The Figure 6 and

Figure 7 show how the LOG and LST files can be written back to SAS Drug Development through a relative path. These metadata locations can be independent of one another. For example, Figure 6 illustrates the LOG file being written to a `'./outlogs'` folder, while Figure 7 shows the LST file being written to `'./outlisting'` folder.

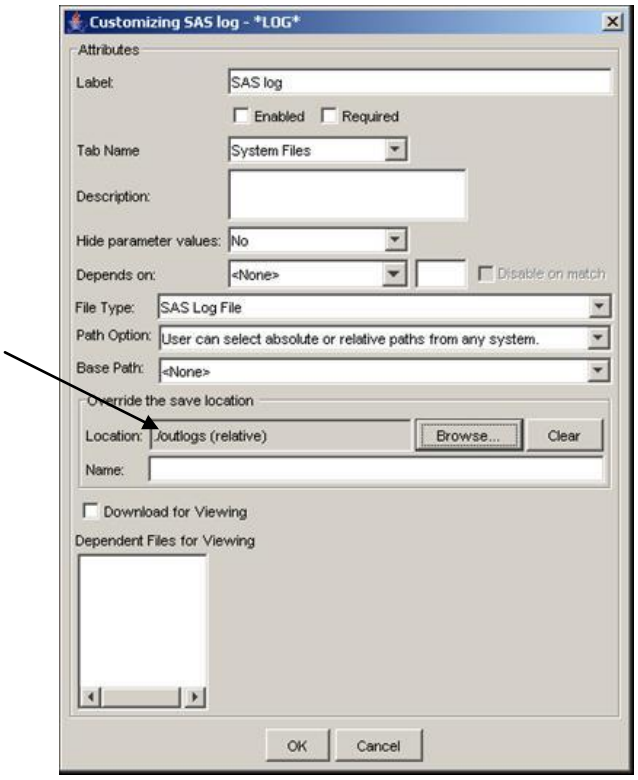

Figure 6: LOG File Written to the './outlogs'Folder

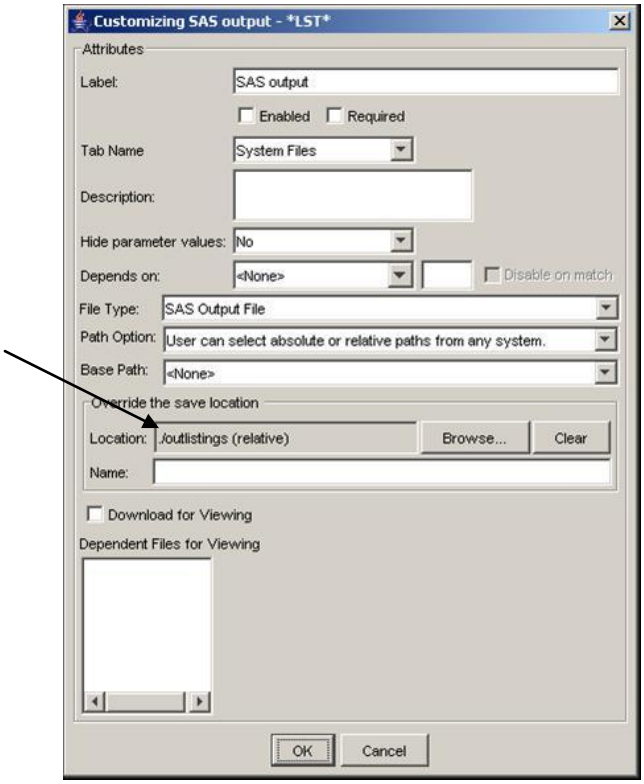

Figure 7: LST File Written to the './outlisting' Folder

INCLUDE METADATA

After completing the LOG and LST file metadata setup in the sample.sas code, complete the following steps:

1. Save this Process Editor program to a local destination. Once saved, opening the file in the SAS System Viewer (or any text editor) reveals the following code.

```
%include &setup;

/* rest of SAS processing code */
/******PACMAN****** DO NOT EDIT BELOW THIS LINE ******PACMAN******/
/*<?xml version="1.0" encoding="UTF-8"?>*/
/*<process sessionid="92218c:12116ee7143:5a73" sddversion="3.4" cdvoption="N" parseroption="B">*/
/* <parameters>*/
/*  <parameter userdefined="S" obfuscate="N" id="&star;LOG&star;" canlinktobasepath="Y" protect="N" label="SAS log" systemtype="&star;LOG&st
/*   type="LOGFILE" autolaunch="N" filetype="LOG" tabname="System Files">*/
/*   <target rootname="" extension="log">*/
/*    <folder system="RELATIVE" source="RELATIVE" displayname="outlogs" id="outlogs" itemtype="Container" fileinfoversion="3.0">*/
/*    </folder>*/
/*   </target>*/
/*   <description>*/
/*   </description>*/
/*  </parameter>*/
/*  <parameter userdefined="S" obfuscate="N" id="&star;LST&star;" canlinktobasepath="Y" protect="N" label="SAS output" systemtype="&star;LST
/*   type="LSTFILE" autolaunch="N" filetype="LST" tabname="System Files">*/
/*   <target rootname="" extension="lst">*/
/*    <folder system="RELATIVE" source="RELATIVE" displayname="outlistings" id="outlistings" itemtype="Container" fileinfoversion="3.0">*/
/*    </folder>*/
/*   </target>*/
/*   <description>*/
/*   </description>*/
/*  </parameter>*/
/*  <parameter dependsaction="ENABLE" obfuscate="N" systemtype="SDDPARMS" label="Process parameter values" tabname="System Files" baseoption
/*   processid="P5" required="N" enable="N" resolution="INPUT" type="PARMFILE">*/
/*  </parameter>*/
/*  <parameter dependsaction="ENABLE" obfuscate="N" systemtype="&star;PGM&star;" label="SAS program" tabname="System Files" baseoption="A" a
/*   processid="P5" required="N" enable="N" resolution="INPUT" type="PGMFILE">*/
/*  </parameter>*/
/*  <parameter dependsaction="DISABLE" obfuscate="N" label="Input process" tabname="Parameters" baseoption="A" advanced="N" order="5" id="SE
/*   resolution="INPUT" type="INPROCESS">*/
/*   <default>*/
/*    <file system="RELATIVE" source="RELATIVE" displayname="setup.sas" id="setup.sas" itemtype="Item" type="sas" fileinfoversion="3.0">*/
/*    </file>*/
/*   </default>*/
/*  </parameter>*/
/*  <parameter id="REST_AUTO" resolution="INTERNAL" type="TEXT" order="7">*/
/*  </parameter>*/
/* </parameters>*/
/*</process>*/
/**/
/******PACMAN****************************************PACMAN******/
```

Figure 8: SAS System Viewer Code

Note: The SAS comment block of XML above is built using the metadata for the LOG and LST using the Process Editor's Parameter User Interface (second method described in the Reverse Engineering section above).

2. Highlight the comment block starting from the line:

```
/******PACMAN****** DO NOT EDIT BELOW THIS LINE ******PACMAN******/
```

3. Finish highlighting at the bottom of the page:

```
/******PACMAN****************************************PACMAN******/
```

4. Copy the highlighted information to the paste buffer. Open the local piece of SAS code with the modified selection of the `%include` file and paste this SAS comment block to the bottom of the program and save the program.

This completes the customizations enabling this code to function in both PC SAS and in SAS Drug Development. It can be used as a template by just altering the SAS code after the `%setup macro` (the code that define which `%include` to use) and no not alter any of the comment block at the bottom of the program (XML metadata).

Assumptions: This code will always function provided that the SAS Drug Development hierarchy is designed consistently across projects, with the same named folders holding the necessary information for input and output location(s).

RUNNING PROCESS EDITOR CODE OUTSIDE OF SAS DRUG DEVELOPMENT

The same concepts covered in converting PC SAS code to run in the Process Editor can be used when removing code from SAS Drug Development and running it as a standalone program in PC SAS. In order for this to work, the Process Editor programs must be developed utilizing the `%include` functionality (input process). Writing programs this way provides the ability to programmatically extract these Process Editor programs and modify the `%include` statement allowing a PC `%include` file to be read. Only this one line of the program is altered; the remaining program remains untouched. Saving the unaltered Process Editor program locally and using a "DIFF" utility to compare it to the altered program proves that only the one line of code is altered. Therefore, if the original program is validated in SAS Drug Development, then, it stands to reason, when provided with the correct inputs, the converted to PC SAS code creates the same output as the program in SAS Drug Development.

API SAS MACROS

The SAS Drug Development API SAS macros enable you to use familiar SAS syntax to make calls to SAS Drug Development. These macros run within the Process Editor or can be installed locally in conjunction with the SAS Drug Development API and used within PC SAS. The code shown in %do `i = 1 %to &loop;`

```
%SASDRUGDEV_CREATELOCALFILE(LOCALPATH=&workpath\&&namer&i, SDDPATH=&logdir/&&namer&i);

filename outpgm "C:\temp_m\FDA\code\&&namer&i";

data _null_;
    file outpgm;
    filename temmp "&workpath\&&namer&i";
    infile temmp lrecl=3260;
        input;
        if index(_infile_, '%include &startup;') > 0 then
       _infile_ = %str('%include "C:\temp_m\FDA\code\startup.sas";');
      put _infile_;
run;
%end;

%mend;
%get_logs;

%sasdrugdev_logout;
```

Figure 9 below can be run in PC SAS.

```
/*****************************************************************************
the local folder C:\temp_m\FDA\code needs to exist or the code need to be modified
*****************************************************************************/

%let LogDIR=%str(/FULL_QUALIFIED_PATH_TO_LOG_LOCATION_IN_SDD);

options nodate;

%sasdrugdev_login(url=%str(https://XXXX.sas.com/sddremote),
           sdduserid=%str(XXXXXX), sddpassword=%str(XXXXXXX));

%sasdrugdev_getobjects(sddpath=&LogDIR,DSNAME=work.log_objs);

data temp;
  set log_objs;
  retain count 0;
  count = count + 1;
  call symput('namer'||left(count), strip(name));
  call symput('loop',count);
run;

%macro get_logs;

data work._null_;
    attrib workpath length=$200;
    workpath=pathname("work");
    call symput("workPath", trim(left(workpath)));
    stop;
  run;
```

```
%do i = 1 %to &loop;
%SASDRUGDEV_CREATELOCALFILE(LOCALPATH=&workpath\&&namer&i, SDDPATH=&logdir/&&namer&i);

filename outpgm "C:\temp_m\FDA\code\&&namer&i";

data _null_;
    file outpgm;
    filename temmp "&workpath\&&namer&i";
    infile temmp lrecl=3260;
        input;
        if index(_infile_, '%include &startup;') > 0 then
       _infile_ = %str('%include "C:\temp_m\FDA\code\startup.sas";');
      put _infile_;
run;
%end;

%mend;
%get_logs;

%sasdrugdev_logout;
```

Figure 9: Code Run in PC SAS

When this SAS code is submitted, any SAS program that is in the SAS Drug Development hierarchy (`/FULL_QUALIFIED_PATH_TO_LOG_LOCATION_IN_SDD`) is converted to run in the PC environment and stored in the `C:\FDA\code` folder locally. These local programs now look for the `%include` file in the "`C:\temp_m\FDA\code\`" folder with the name "`startup.sas`".

This is illustrated in Figure 10 below:

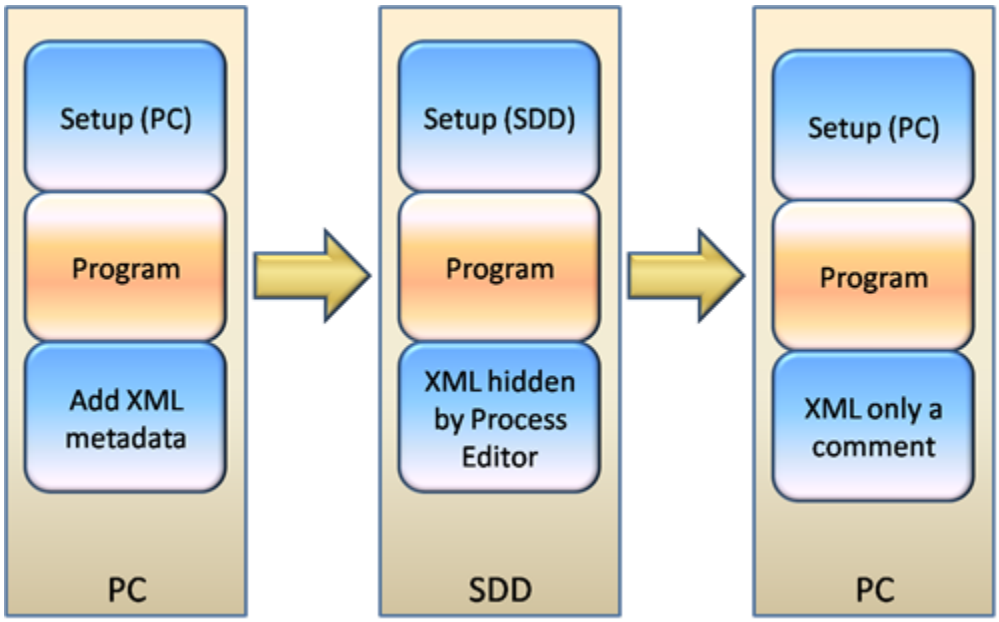

Figure 10: Process to Convert SAS Process to Run in the PC Environment

CONCLUSION

In order to make processes efficient the ability to promote validated code to production environments without alteration is essential. Validated, reusable, and portable programs are some fundamental requirements of SAS code developed in the Pharmaceutical industry today. SAS Drug Development helps meet these requirements. The goal of this paper was to show how SAS code developed outside of SAS Drug Development can be utilized within the system and, if designed with this goal in mind, the transition of this code into SAS Drug Development can be seamless.

REFERENCES

SAS Drug Development Users Guide V3.5
SAS Drug Development API Macros Users Guide V3.5

ACKNOWLEDGMENTS

Nick Ronca and John Anthony Shire Pharmaceuticals for help working out the methodology and construct of SAS programs so that these methods could be proven and meet validation expectations.

CONTACT INFORMATION

Ben Bocchicchio
SAS Institute
SAS Campus Drive
Cary, NC 27513
Office : (919) 531-3704
Fax: (919) 531-0700
Email: ben.bocchichio@sas.com
Web: http://www.sas.com