PP08: Production SAS Code Polishing Concepts for Regulatory Submissions

Nori Oharu (Norihiko.Oharu@pfizer.com), **Ravi Mokkarala** (Ravi.mokkarala@pfizer.com)

Statistical Programming and Analysis, Global Product Development, Pfizer, Inc.

Introduction:

Executable SAS programs in ASCII format which create primary and secondly endpoints for a submission pivotal study are often requested as part of regulatory submission. Pharmaceutical companies which utilize standard data reporting system likely use sas codes which use multiple macros, utilities, parameters and dependencies. This setup could be challenging to create **executable sas programs** with all associated dependencie printed with SAS ***MFILE*** and ***MPRINT*** options. As part of the effort in standardized reporting environment in one of our submissions, we needed to apply **concept of code polishing** which makes the SAS programs **cleaner, easier to read and executable** for reviewers outside of the specific reporting environment with **minimum to no adjustment** with built-in QC process.

Concept:

While executable primary and secondly efficacy sas codes were agreed to be included as part of our submission package, timelines were discussed for final reporting. Due to the complexities of standardized sas program set up, adjusting SAS files executed with ***MFILE*** and ***MPRINT*** takes **significant amount of time with risk of human errors**. Manual adjustments are required each time after SAS codes are executed and will contain large number of lines printed from all associated files called from the main SAS code and will not be easily executed in the different reporting environment.

To overcome challenges, **metafile driven code polishing concepts** were introduced that produce user friendly executable sas code which can be executed in any reporting environment which reduces time and errors. The process also includes creation of .txt files and Validation steps.

Ex. *abc.sas:* a sas program to create a table.
Execute abc.sas with SAS options *MPRINT* and *MFILE* and create file with all macros, dependencies printed within the single file (.sas) created.
Set up metafiles: Identify and add what you want to "**polish**" on the .sas file. (ex. Comments from associated macros included) from *abc.sas* which is not relevant to create the table.

Metafile set up…

Metafile should be maintained by programmer by looking at the .sas file…

```
proc sort data=_bydat1;
by _datasrt ;
run;
proc print data=_bydat1;
title 'print of _bydat1';
run;
proc sort data = _data1 out = _data1;
by _datasrt ;
run;
options MPRINT SYMBOLGEN MLOGIC;
```

Metafile should include what you want to "polish" from the MPRINT file (.sas).

Metafile: **polish.dat**

- pclear^proc print
- options MPRINT.*;^

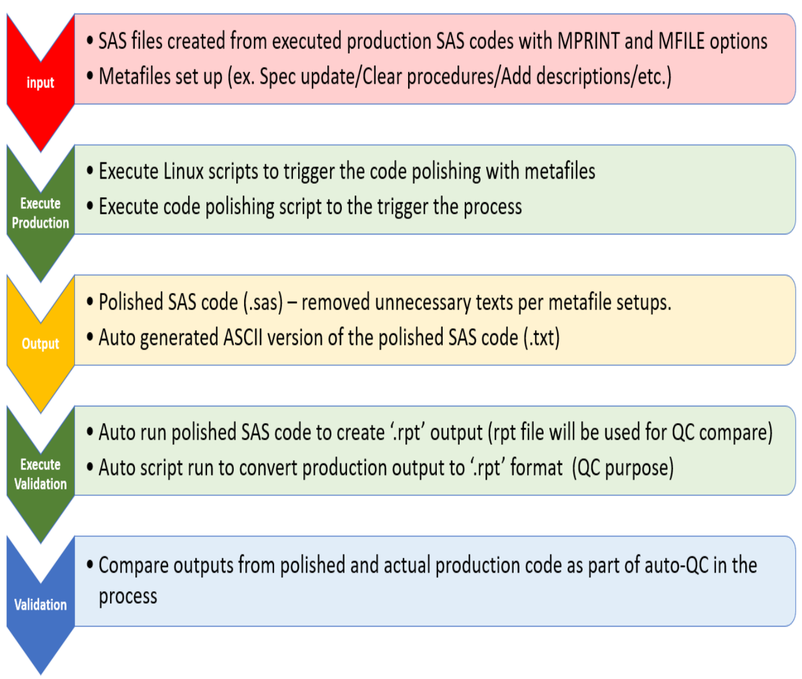
Note: Remove SAS procedure and options which are not relevant in the production code and stops at "run;".

Execution:

After initial metafile set up it is just one command away

➢ doplish.sh <SAS filename – t/f/a>

Process flow:



Sample outputs (before and after):

```
**** exit if not running a DP algorithm ************************************;
**** there were some previous study specific algorithms that had the *********;
**** same parameters as this program uses. Building in a method to skip *****;
**** this pgm even when the tot has DSIN and SUBSET or PAGEBY ****************;
**** look for the new parameter naming (G_) *********************************;
;
;
;
;
**** See if the this presentation is using subsetting or stratification ******;
*******************************************************************************;
* SPECIFICATION 1 - CHECK INPUT DATASET *;
* 1) Check if reporting dataset is present, if not call util_bail_out *;
* utility else continue with code. *;
*******************************************************************************;
;
*******************************************************************************;
* SPECIFICATION 1 - a) Initialize setup and display ISAM parameters *;
* b) Initialize fmtgen and treatmapgen parameters *;
* some of these parameters are optional for some of the ToTs. Any of listed *;
* parameters can be added to ToTs to trigger the functionality specific to *;
* a parameter *;
*******************************************************************************;
*******************************************************************************;
* SPECIFICATION 2 - apply G_ADSL_SUBSET and G_A_SUBSET. *;
* create a variable analysis_subset that flags Y for all *;
* g_a_dsin records. *;
* create g_dictver. *;
* set G_PAEPGNDR to AEPGNDR when it is populated on ADAE *;
*******************************************************************************;
options MPRINT MLOGIC SYMBOLGEN;
options MPRINT SYMBOLGEN MLOGIC;
;
data g_a_dsin;
set datvprot.adtte;
analysis_subset='Y';
run;
data g_adsl_dsin;
set datvprot.adsl;
where ITTFL='Y';
run;
data g_a_dsin;
set g_a_dsin;
where ARMCD ne ' ' and PARAMN = 212;
run;
```

```
/*****************************************************************/
Table 14.2.2.7.1:Summary of iDFS Excluding Second Primary Invasive
Non-Breast Cancers - Sensitivity Analysis 2 - ITT Analysis Set
TOT name: adtte_idfs_ns014.tot
Input datasets: adsl adtte adtte_superset adtte_superset.dat
Change paths according to XPT or datasets, and output locations
/*****************************************************************/

/* This is smart polished code */
options mprint mlogic symbolgen;
libname sdtm 'tabulation data path';
libname adam 'analysis data path';

data g_a_dsin;
set adam.adtte;
analysis_subset='Y';
run;
data g_adsl_dsin;
set datvprot.adsl;
where ITTFL='Y';
run;
data g_a_dsin;
set g_a_dsin;
where ARMCD ne ' ' and PARAMN = 212;
run;
```

```
proc sort data=_bydat1;
by _datasrt ;
run;
proc print data=_bydat1;
title 'print of _bydat1';
run;
proc sort data = _data1 out = _data1;
by _datasrt ;
run;
options MPRINT SYMBOLGEN MLOGIC;
```

```
proc sort data=_bydat1;
by _datasrt ;
run;
<
<
<
proc sort data = _data1 out = _data1;
by _datasrt ;
run;
|
```

Conclusions:

The process of "**code polishing**" can be further simplified in our production environment; however, the concept of "code polishing" is end to end process which can be developed and maintained throughout the programming activities. This includes ASCII file creation and Validation steps with one execution of Linux script which **reduce programming cycle time and eliminate human errors** within the tight submission timelines.

Acknowledgment:

Sharad Chettri & Pfizer FSP programming team for contributions to Code Polishing script development

DocUUID : 0038859c-a04f-421f-924f-3ed218174289