

Project Automation and Tracking Using SAS

Rajesh Lal, COMSYS, Portage, MI

ABSTRACT

During the life cycle of a clinical trial study/project, there are various tasks that are done manually, ranging from project setup to tracking of project status, and project wind-up. Manual tasks are always time and effort intensive and prone to errors, especially when the number of programs and outputs to be created in the project is large and timelines are tight. By automation of some of these tasks, timely production and validation of standardized programs and outputs can be ensured.

This paper describes the design of a utility that is implemented in SAS®, automates the project startup related activities, keeps track of validation comparison status, status from validation documents, re-validation requirements, version control status, SAS log check status, program readability status, overall validation status and performs project wind-up activities. The utility helps ensure that a project team always has an up-to-date picture of validation status during the life cycle of a project, is regulatory compliant for computer system validation and prepared for internal or external audits.

INTRODUCTION

A clinical trial study can have hundreds of SAS programs creating datasets, tables, listings and graphs as outputs. In a fast-paced production environment, where timelines are short and last minute change requests are frequent, quick setup of the project, timely and accurate tracking of project status and health, and quick wind-up of the project are crucial to the timely delivery of the outputs for regulatory submissions.

By using Program Index (PI), which is a Microsoft Excel® file, to store the list of all the SAS programs required for a clinical trial study/project and other related information at a central place, many tasks can be automated by reading in the information from the PI.

This paper introduces a SAS utility that reads in the PI, performs the project startup related activities to eliminate manual entry of the information that is already present in the PI, keeps track of validation comparison status, status from validation documents, re-validation requirements, version control status, SAS log check status, program readability status, overall validation status and performs project wind-up activities.

The utility helps ensure that a project team always has an up-to-date picture of validation status during the life cycle of a project, is regulatory compliant for computer system validation and prepared for internal or external audits.

The utility reads in the PI into a SAS dataset, performs the required operations, creates or updates the variables in the SAS dataset for various status information and writes the SAS dataset back to MS Excel PI file. This utility is comparable to a utility called "PI Manager" developed by Tony Chang & Dana Soloff, Amgen Inc., which has a subset of features, is implemented in java, and therefore is not easily editable/customizable by a SAS programmer.

OVERVIEW

The utility is implemented for unix SAS 9.2, but can be ported to unix SAS 9.1.3 or PC SAS 9.1.3/9.2 with some limitations, as described in 'Design considerations and implementation' section. The following is a list of features of the utility:

Automation of project start-up related activities:

1. Create standard directory structure for the project
2. Create empty SAS program files with header information, or template programs for specific type of TLGs.
3. Create a shell script or batch file for the project
4. Create empty validation documents with production program/validation program/ validation document information and program complexity dependent checklist of validation activities

Tracking of project status:

5. Updating the date/time stamp of SAS programs/validation documents #
6. Getting validation status from validation documents #
7. Identifying re-validation requirements #
8. Summarizing overall programming status #
9. Version control status (RCS and TeamWare compatible)
10. Summary of SAS log errors, warnings and other Notes ⁶
11. Fetching validation comparison status from SAS list file of the validation program
12. Program readability

Project wind-up activities:

13. Check-in the programs, make the outputs read-only
14. Redundant programs/files in production/validation location

Figure 1 below shows a flow chart that describes how the utility performs its operations.

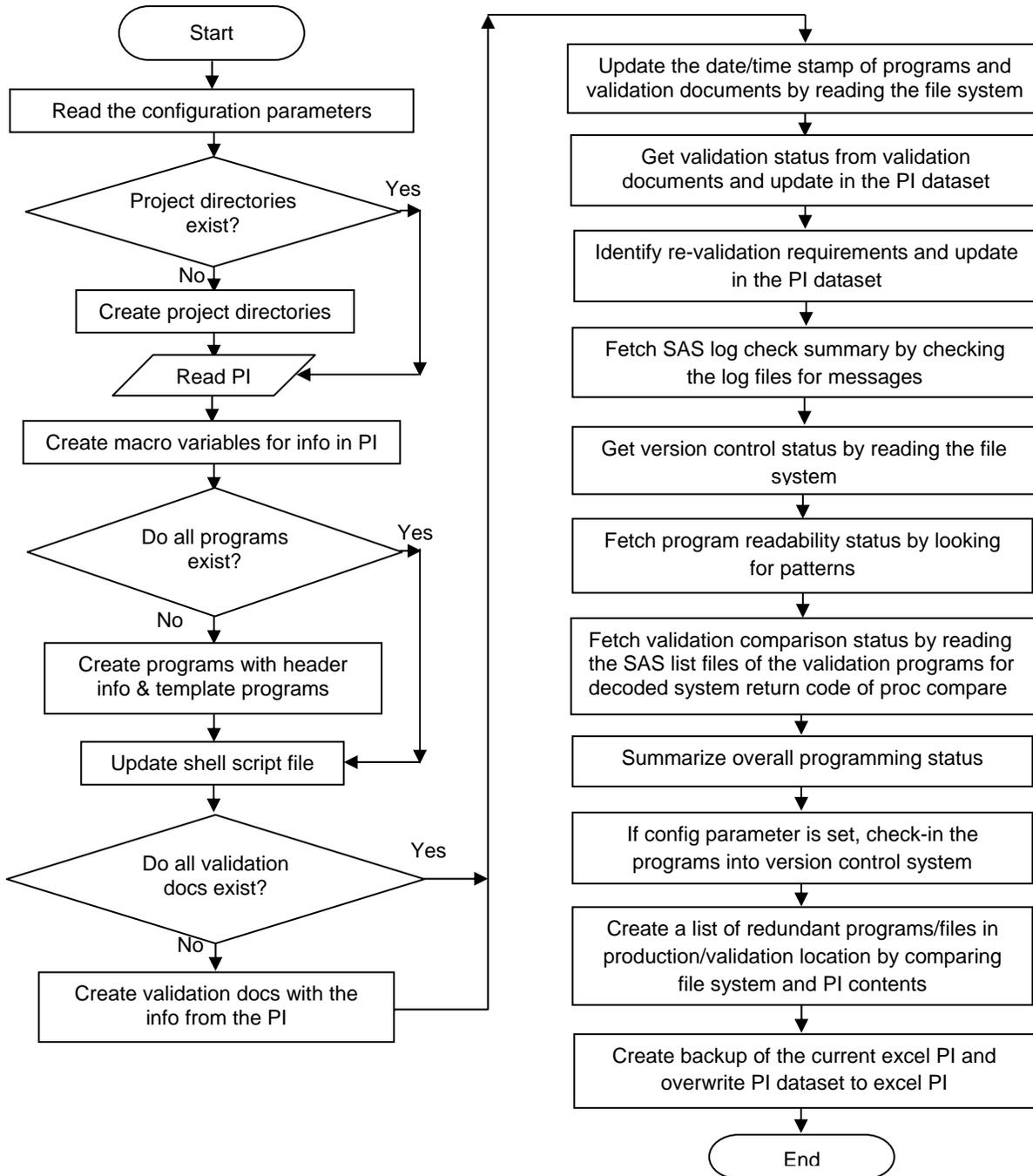


Figure 1. Flow chart

PROGRAM INDEX FILE

Program Index file is a Microsoft Excel file, which contains the list of all SAS programs required for a clinical study/project, program author names, template names, program complexity, program modified date, version control status, SAS log check status, Program readability status, output filenames, validation program name, validation program author names, validation program modified date, validation document name, last validation date, comparison results, validation document status and overall status. Table 1 below shows a PI with sample data:

SR_NO	PGM_TYP	PGM_NAME	PGM_AUTHOR	TEMP_LATE_NAME	COMPLE_XITY	PGM_LAST_MOD_DATE	VERSION_CONT_RL_STATUS	LOG_CHECK_STATUS	PGM_READABILITY	OUTPUT_FILE_NAME	VALIDATION_PGM	VALIDATION_PGM_AUTHOR	VALIDATION_PGM_LAST_MOD_DATE	VALIDATION_DOCUMENT	LAST_VALIDATION_DATE	COMPARISON_RESULT	VALIDATION_DOCUMENT_STATUS	OVERALL_VALIDATION_STATUS
1	S	dm.sas	John		E	29-Nov-2010 09:29	Not chkd in	Errors, Warnings, Notes	Dead code, Bad lindent	dm.sas7bdat	v_dm.sas	Harry	30-Nov-2010 10:12	v_dm.xls	30-Nov-2010 11:20	Matching	Pass	Pass
2	S	ae.sas	Harry		M	30-Nov-2010 13:51	Chkd in	Warnings, Notes	OK	ae.sas7bdat	v_ae.sas	John	30-Nov-2010 14:21	v_ae.xls	30-Nov-2010 14:25	BY variables do not match	Fail	Fail
		...																
3	T	t_ae.sas	Harry	aetmp.sas	M	30-Nov-2010 14:59	Chkd in	Clean	Insufficient comments	t_ae_saf.rtf	v_t_ae.sas	John	29-Nov-2010 11:22	v_t_ae.xls	29-Nov-2010 11:26	Matching	Pass	Re-validate
4	T	t_ae.sas	Harry	aetmp.sas	M	30-Nov-2010 14:59	Chkd in	Clean	Insufficient comments	t_sae_saf.rtf	v_t_ae.sas	John	29-Nov-2010 11:22	v_t_ae.xls	29-Nov-2010 11:26	A value comparison was unequal	Pass	Re-validate
5	G	g_km.sas	John		C	30-Nov-2010 13:45	Chkd out	Warnings	OK	g_km.cgm	v_g_km.sas	Harry	01-Dec-2010 18:45	v_g_km.xls	30-Nov-2010 13:49	Manual comparison	Pass	Pass

Table 1. Sample PI

UTILITY FEATURES

AUTOMATION OF PROJECT START-UP RELATED ACTIVITIES:

1. CREATE STANDARD DIRECTORY STRUCTURE FOR THE PROJECT

The utility creates standard directory structure for a project by running operating system commands, depending upon the locations specified in configuration parameters. Below is a SAS macro that checks for the existence of a directory and creates the directory if it does not exist.

```

/* configuration parameters */
%let sdtm = /compoundA/studyX/sdtm/programs/;
%let sdtm_out = /compoundA/studyX/sdtm/output/;
%let sdtm_test = /compoundA/studyX/sdtm/validation/;
...
%macro mk_dir(dir=) ;
  %local rc fileref ;
  %let rc = %sysfunc(filename(fileref,&dir)) ;
  %if %sysfunc(fexist(&fileref)) %then
    %put Note: the directory "&dir" exists ;
  %else
  %do ;
    %sysexec mkdir -p &dir ;
    %put Note: the directory "&dir" has been created. ;
  %end ;
  %let rc=%sysfunc(filename(fileref)) ;
%mend mk_dir ;

%mk_dir(dir=&sdtm);
%mk_dir(dir=&sdtm_out);
%mk_dir(dir=&sdtm_test);

/* repeat for other project locations */

```

2. CREATE EMPTY SAS PROGRAM FILES WITH HEADER INFORMATION, OR TEMPLATE PROGRAMS FOR SPECIFIC TYPE OF TLGS

The utility creates blank programs with standard header, populated with information such as program name & location, output name & location and the name of the programmer. If there is a template program for a particular type of program, the template

is appended to the standard header, so that the programmer can quickly modify certain things in the template and get the program up and running, e.g. a template program for creating AE tables, using a standard AE macro can be used for creation of the standard AE tables. Below is a piece of SAS code that creates the standard header and appends any template programs.

```

/* read in PI file into a SAS dataset, pi_ds, using proc import. Please refer to the 'Design
considerations and implementation' section for details on reading from/writing to Excel files
*/
...
/* create 1 observation per SAS program, by appending the name of multiple outputs */
...
/* create macro variables for program names & locations, outputs & locations, programmer
names */
...
/* create standard header using macro variables created above */
%do i=1 %to &pgm_cnt;
  %if not %sysfunc(fileexist("&pgm_path&i.&pgm_name&i")) %then %do;
    data _null_;
      file "&pgm_path&i.&pgm_name&i" notitle nofootnote;
      put "/*****";
      put "Program Name : &pgm_name&i";
      put "Program Path : &pgm_path&i";
      Put "Output      : &out_path&i.&out_names&i";
      Put "Programmer   : &pgm_author&i";
      put "*****/";
      put "%include 'setup.sas'";
      put ;
    run;
    %if &tmpl_name&i ne %str() %then %do;
      %if %sysfunc(fileexist("&tmpl_path.&tmpl_name&i")) %then %do;
        data _null_;
          infile "&tmpl_path.&tmpl_name&i";
          file "&pgm_path&i.&pgm_name&i" mod notitle nofootnote;
          input;
          put _infile_;
        run;
      %end;
    %else %do;
      %put The template file "&tmpl_path.&tmpl_name&i" does not exist;
    %end;
  %end;
%end;

```

Figure 2 shows the standard program header generated by above SAS code. Please note that the template program has been appended to the standard header.

```

1  /*****
2  Program Name : t_ae.sas
3  Program Path : /compoundA/studyX/table/programs/
4  Output      : /compoundA/studyX/table/output/t_ae_saf.rtf, t_sae_saf.rtf
5  Programmer   : Harry
6  *****/
7  %include 'setup.sas'
8
9  /*****
10
11 THIS IS AE PROGRAM TEMPLATE FROM /compoundA/studyX/templates/aetmp.sas
12
13 CALL AE STANDARD MACRO
14
15 *****/
16

```

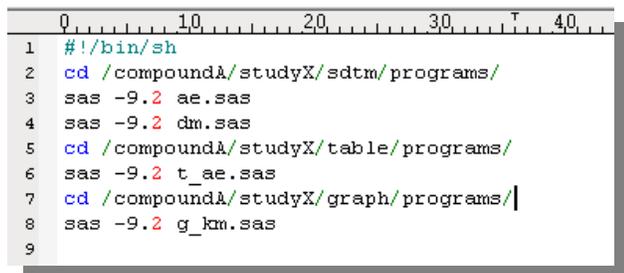
Figure 2. Standard header and appended template generated by above SAS code

3. CREATE A SHELL SCRIPT OR BATCH FILE FOR THE PROJECT

The utility creates a unix shell script file or a batch file for all the programs in the project, which can be scheduled to run at specific intervals to keep the project outputs and status up to date.

```
%if not %sysfunc(fileexist("&sdtm.run_all.sh")) %then %do;
data _null_;
  set pi_ds1;
  where pgm_type = "S";
  file "&sdtm.run_all.sh";
  if _n_ eq 1 then do;
    put "#!/bin/sh";
    put "cd &sdtm";
  end;
  put "sas -9.2 " pgm_name;
run;
data _null_;
  set pi_ds1;
  where pgm_type = "T";
  file "&sdtm.run_all.sh" mod;
  if _n_ eq 1 then do;
    put "cd &table";
  end;
  put "sas -9.2 " pgm_name;
run;
/* repeat the above step for listings and graphs */
%end;
```

Figure 3 shows the shell script file generated by executing the above SAS code.



```
0 10 20 30 40
1 #!/bin/sh
2 cd /compoundA/studyX/sdtm/programs/
3 sas -9.2 ae.sas
4 sas -9.2 dm.sas
5 cd /compoundA/studyX/table/programs/
6 sas -9.2 t_ae.sas
7 cd /compoundA/studyX/graph/programs/
8 sas -9.2 g_km.sas
9
```

Figure 3. Shell script file generated by above SAS code

4. CREATE EMPTY VALIDATION DOCUMENTS WITH PRODUCTION PROGRAM/VALIDATION PROGRAM/ VALIDATION DOCUMENT INFORMATION AND PROGRAM COMPLEXITY DEPENDENT CHECKLIST OF VALIDATION ACTIVITIES

The utility creates validation documents (Microsoft Excel) with production and validation program name and location information populated, so that manual entry of this information is not needed. Validation programmer should complete the validation by following the validation checklist in the validation document and should mark the status of the checklist item.

Once the outputs are validated, the validation programmer should update the validation status in the validation document. Please refer to the 'Design considerations and implementation' section for details on reading from/writing to Excel files. The piece of SAS code below shows the creation of the validation document. Please note that the checklist of activities to be performed during validation is customized based upon the complexity of the program, which is read from the PI.

```
%do i=1 %to &pgm_cnt;
/* create validation documents using macro variables created */
  %if not %sysfunc(fileexist("&&test_path&i.&&test_doc&i")) %then %do;
    data temp;
      length a b $200;
      a="Validation Status"; b=" "; output;
      a="Validation Document Name"; b="&&test_path&i.&&test_doc&i"; output;
      a="Program Name"; b="&&pgm_path&i.&&pgm_name&i"; output;
      a="Output          "; b="&&out_path&i.&&out_names&i"; output;
```

```

a="Programmer "; b="&&pgm_author&i"; output;
a="Validation Programmer"; b="&&val_author&i"; output;
a=" "; b=" "; output;
a=" "; b=" "; output;
a="Check list of validation activities "; b="Activity Status (Pass/Fail)"; output;
%if &&pgm_cmpx&i = E %then %do;
/* checklist for 'Easy' programs. Repeat for 'Medium' and 'Complex' programs */
a="Check list item 1 "; b=" "; output;
a="Check list item 2 "; b=" "; output;
a="Check list item 3 "; b=" "; output;
a="Check list item 4 "; b=" "; output;
%end;
label a="Program Validation Document" b="Template Version 0.1";
run;
proc export data = temp
outfile= "&&test_path&i.&&test_doc&i"
dbms=xls label replace;
run;
%end;
%end;

```

Figure 4 shows the validation document, v_t_ae.xls, as viewed in Microsoft Excel, generated by the SAS code above for program t_ae.sas.

	A	B	C	D
1	Program Validation Document	Template Version 0.1		
2	Validation Status			
3	Validation Document Name	/compoundA/studyX/table/validation/v_t_ae.xls		
4	Program Name	/compoundA/studyX/table/programs/t_ae.sas		
5	Output	/compoundA/studyX/table/output/t_sae_saf.rtf, t_ae_saf.rtf		
6	Programmer	Harry		
7	Validation Programmer	John		
8				
9				
10	Check list of validation activities	Activity Status (Pass/Fail)		
11	Check list item 1			
12	Check list item 2			
13	Check list item 3			
14	Check list item 4			
15				

Figure 4. Validation document generated by SAS code

TRACKING OF PROJECT STATUS:

5. UPDATING THE DATE/TIME STAMP OF SAS PROGRAMS/VALIDATION DOCUMENTS

The utility collects last modified system date and time for a SAS program, an output file, and a validation document specified in the Program Index file, if the file exists. Please refer to the 'Design considerations and implementation' section for details on fetching file modified date and time. The operating system dependent SAS code below shows the method to fetch this information from file system for Microsoft Windows® and unix based systems.

```

/*updating the date/time stamp of sas programs and validation documents*/
%macro upd_dttm(mrgvar=,dtvar=,loc=,ext=sas);
/* for Microsoft Windows based systems */
%if &SYSSCP = %str(WIN) %then %do;
filename tmpfl pipe "dir &loc.*.&ext";
data fs1(keep=fnm dttm rename=(fnm=&mrgvar));
infile tmpfl firstobs=6;
length fnm $200 dt $10 tm $5;
input dt $10. @;
if missing(dt) then delete;
else input tm $ ampm $ fsz $ fnm $;
dttm = dhms(input(dt,mmddy10.), hour(input(cats(tm,ampm),time8.)),
minute(input(cats(tm,ampm),time8.)),0);

```

```

        format dttm datetime16.;
run;
%end;

/* for unix based systems */
%else %if &SYSSCP = %str(SUN 64) %then %do;
    filename tmpfl pipe "ls -l --time-style=long-iso &loc.*.&ext";
    data fs1(keep=fnm dttm rename=(fnm=&mrgvar));
        infile tmpfl firstobs=2;
        length fnm $200 permissions usr grp dt fsz $10 tm $5;
        input permissions $ var2 usr $ grp $ fsz $ dt $ tm $ fnm $;
        dttm = dhms(input(dt,yymmdd10.), hour(input(tm,time5.)),
            minute(input(tm,time5.)),0);
        format dttm datetime16.;
    run;
%end;

/* update the file modified date time in PI dataset */
proc sql noprint;
    create table pi_ds_upd as
    select a.*, coalesce(b.dttm,a.&dtvar) as new_dt format=datetime16.
    from pi_ds as a left join fs1 as b
    on a.&mrgvar = b.&mrgvar;
quit;

data pi_ds(drop=new_dt);
    set pi_ds_upd;
    &dtvar = new_dt;
run;
filename tmpfl clear;
%mend upd_dttm;

/* update program modification dates */
%upd_dttm(mrgvar=pgm_name,dtvar=pgm_last_mod_date,loc=&sdtm);
%upd_dttm(mrgvar=pgm_name,dtvar=pgm_last_mod_date,loc=&table);
%upd_dttm(mrgvar=pgm_name,dtvar=pgm_last_mod_date,loc=&listing);
%upd_dttm(mrgvar=pgm_name,dtvar=pgm_last_mod_date,loc=&graph);
/* repeat for validation programs and validation documents */

```

6. GETTING VALIDATION STATUS FROM VALIDATION DOCUMENTS

A validation document is a Microsoft Excel file, which has the details of the program being validated, a checklist of validation activities to be performed, and also an overall "Pass" or "Failed" status field. The validation document is read in using proc import. The validation status field fetched from the validation document is populated against each SAS program.

7. IDENTIFYING RE-VALIDATION REQUIREMENTS

After completion of validation, if production program is modified, this means that a re-validation is needed. This is accomplished by comparing the date time of production program, validation program, and validation document.

8. SUMMARIZING OVERALL PROGRAMMING STATUS

The utility generates an overall validation report for the study. It summarizes total number of outputs and number and percent of outputs for different development status. Table 2 below is an example of the summary report:

Project Status Report:	
Total Number of Output Files:	100
Number of Files in Programming:	10 (10%)
Number of Files under Validation	1 (1%)
Number of Files Validated:	86 (86%)
Number of Files Need Re-Validation:	8 (8%)
Number of Files Validation Passed:	60 (60%)
Number of Files Validation Failed:	15 (15%)

Table 2: Sample overall programming status

9. VERSION CONTROL STATUS (RCS AND TEAMWARE COMPATIBLE)

The utility fetches the version control status of each program and populates it in the PI. This feature is available on the unix platform, using RCS or TeamWare version control systems. Other version control systems which use a sequential file in a fixed name sub-folder can also be tracked. The following criteria are used for status:

- a) Not version controlled – the sequential file used by version control system does not exist in the expected sub-folder
- b) Checked in – the sequential file exists and the main file is read-only.
- c) Checked out – the sequential file exists and the main file is writable.

This information is updated in the “Version Control Status” column for each program in the PI.

Please see 'Design considerations and implementation' section for details of fetching file read-only attribute.

10. SUMMARY OF SAS LOG ERRORS, WARNINGS AND OTHER NOTES ⁶

For each program listed in the PI, the utility scans the SAS log file for errors, warnings and other Notes. Summary of this information is updated in the PI under column “Log Check Status” for each program. Depending upon the value of configuration parameter email_on, an email is sent out to the programmer of the program. Table 3 below shows the criteria used:

email_on value	An email is sent out to the programmer if:
0	Never
1	There are errors in the log
2	There are errors and/or warnings in the log
3	There are errors and/or warnings and/or notes in the log

Table 3. Parameter values and criteria for posting an email

11. FETCHING VALIDATION COMPARISON STATUS FROM SAS LIST FILE OF THE VALIDATION PROGRAM

If the validation program compares the production version of the output with the validation version using proc compare and prints the decoded value of system return code in the SAS list file (.lst) along with a uniquely identifiable text string, the utility can fetch the proc compare result from the list file and populate against each program in the PI under “Comparison Result” column. If the expected text string is not found then it is assumed that a manual comparison is being done.

For ease of implementation, the proc compare step and printing of the system return code can be combined in a macro. For comparing tables and listings in rich text file format (.rtf), these files can be converted to SAS datasets. The piece of SAS code below shows how a validation program should compare the validation output with production output and print the decoded values of system return code in the SAS list file.

```

/* compare */
proc compare base=prod compare=qc ;
run;
/* store the system return code in a macro variable */
%let rc=&sysinfo;
/* print the decoded values in the list file along with output names */
data _null_;
  file print notitles nofootnotes;
  put "Proc compare RC from SYSINFO for &out_name: Start";
  if &rc='.....1'b then put 'Data set labels differ';
  if &rc='.....1.'b then put 'Data set types differ';
  if &rc='.....1..'b then put 'Variable has different informat';
  if &rc='.....1...'b then put 'Variable has different format';
  if &rc='.....1....'b then put 'Variable has different length';
  if &rc='.....1.....'b then put 'Variable has different label';
  if &rc='.....1.....'b then put 'Base data obs not in comparison';
  if &rc='.....1.....'b then put 'Comparison data obs not in base';
  if &rc='.....1.....'b then put 'Base data BY group not in comp';
  if &rc='.....1.....'b then put 'Comp data BY group not in base ';
  if &rc='.....1.....'b then put 'Base data variable not in comp';
  if &rc='....1.....'b then put 'Comp data variable not in base ';
  if &rc='...1.....'b then put 'A value comparison was unequal ';
  if &rc='..1.....'b then put 'Conflicting variable types ';
  if &rc='.1.....'b then put 'BY variables do not match ';

```

```

if &rc='1.....'b then put 'Fatal error: comparison not done ';
put "Proc compare RC from SYSINFO for &out_name: End";
run;

```

12. PROGRAM READABILITY

The utility checks to see if there are “enough” comments, proper indentation, and absence of dead code in the programs. Correspondingly, a message is written in the column in the program index. These characteristics of a program are not easily “quantifiable” or “programmatically traceable” and are indicative only.

- a) “Enough” comments: Number of comments in a program is counted and compared to total lines of code in the program. Using a threshold specified in configuration parameter “comnt_thr”, a decision is made and appropriate status is written to the “Program Readability” column in the PI. The SAS code below counts the number of single and multiline comments using pearl regular expressions.

```

*** read in the sas program file;
data pgm;
  infile "&&pgm_path&i.&&pgm_name&i" trunccover;
  length txt $500;
  input;
  txt = _infile_;
run;
*** count and remove comments;
data pgm1;
  set pgm;
  *** cmnts = number of comments in the program,
      cntdstar = 1, if in the current record, there is a comment started with a '*'
          that continues on the next record.
      cntdslash = 1, if in the current record, there is a comment started with a '/'
          that continues on the next record.;
  retain cmnts 0 cntdstar cntdslash 0;
  txt_ = txt;
  *** comments on a single line;
  if not cntdslash and (prxmatch("/(\\;)(\\s)*(\\*)+[^;]*(\\;)/",txt)
    or
    prxmatch("/^(\\s)*(\\*)+[^;]*(\\;)/",txt)
    or
    prxmatch("/(\\/\\*)+[\w\\W]*(\\*\\//)",txt)) then do;
    txt_ = prxchange("s/(\\;)(\\s)*(\\*)+[^;]*(\\;)/ /",-1,txt);
    txt_ = prxchange("s/^(\\s)*(\\*)+[^;]*(\\;)/ /",-1,txt_);
    txt_ = prxchange("s/(\\/\\*)+[\w\\W]*(\\*\\//) /",-1,txt_);
    cmnts = cmnts + 1;
  end;
  *** multiline comments starting with a slash and star;
  else if not cntdslash and prxmatch("/(\\/\\*)+[\w\\W]*/",txt) and not
prxmatch("/(\\/\\*)+[\w\\W]*(\\*\\//)",txt) then do;
    cntdslash = 1;
    txt_ = prxchange("s/(\\/\\*)+[\w\\W]* /",-1,txt);
  end;
  else if cntdslash and not prxmatch("/[\\w\\W]*(\\*\\//)",txt) then do;
    txt_ = " ";
  end;
  else if cntdslash and prxmatch("/[\\w\\W]*(\\*\\//)",txt) then do;
    cntdslash = 0;
    txt_ = prxchange("s/[\\w\\W]*(\\*\\//) /",-1,txt);
    cmnts = cmnts + 1;
  end;
  *** multiline comments starting with a star;
  else if not cntdstar and
    (
      (prxmatch("/(\\;)(\\s)*(\\*)+[^;]*",txt) and

```

```

        not prxmatch("/(\;)(\s)*(\;)+[^\;]*(\;)/",txt))
        or
        (prxmatch("/^(\s)*(\;)+[^\;]*/",txt) and not prxmatch("/^(\s)*(\;)+[^\;]*(\;)/",txt))
    ) then do;
        cntdstar = 1;
        txt_ = prxchange("s/(\;)(\s)*(\;)+[^\;]*/ /",-1,txt);
        txt_ = prxchange("s/^(\s)*(\;)+[^\;]*/ /",-1,txt_);
    end;
else if cntdstar and not prxmatch("/[^\;]*(\;)/",txt) then do;
    txt_ = " ";
end;
else if cntdstar and prxmatch("/[^\;]*(\;)/",txt) then do;
    cntdstar = 0;
    cmnts = cmnts + 1;
    txt_ = prxchange("s/[^\;]*(\;)/ /",-1,txt);
end;
run;

```

- b) Proper indentation: for each data and proc step, indentation is checked in each SAS statement, until the “end” of the group is reached. Appropriate text is written to the “Program Readability” column in the PI. The SAS code below determines the indentation of the first statement after a data or proc statement and sets a variable value to 1 if there are indentation problems. Similar concept can be expanded to cover nested indentation of blocks of SAS code. Please refer to “Future Developments” section for details of other blocks of SAS code that should be considered.

```

data pgm2;
/* txt_ contains the text without any SAS comments */
set pgm1(keep=txt_ where=(not missing(txt_)));
retain in_dtpc blk_ind in_blk 0;
*** replace each tab character with defined number of spaces;
txt_ = prxchange("s/^\t*/ /",-1,txt_);
*** find the indentation of the first statement after the start of the block;
if not blk_ind and in_blk then do;
    blk_ind = prxmatch("/\S/",txt_);
end;
*** mark the start of a data or proc step ;
if prxmatch("/(data |proc )[\w\W]*/",txt_) then do;
    blk_col=prxmatch("/(data |proc )[\w\W]*/",txt_);
    blk_ind = 0;
    if not prxmatch("/(data |proc )[\w\W]*;/",txt_) then do;
        in_dtpc = 1;
    end;
    else do;
        in_blk = 1;
    end;
end;
*** if data or proc statement is on multiple lines, continue to next line;
else if in_dtpc and prxmatch("/[\w\W]*;/",txt_) then do;
    in_dtpc = 0;
    in_blk = 1;
end;
*** end of the data or proc step;
else if prxmatch("/\s*(run|quit)\s*/",txt_) then do;
    in_blk = 0;
    blk_ind = 0;
end;
*** if the indentation of 2nd statement after data or proc statement is not the same as
    1st statement after data or proc step, there may be an indentation problem;
if in_blk and blk_ind then do;
    if blk_ind ne prxmatch("/\S/",txt_) then indt_err = 1;

```

```
end;
run;
```

- c) Dead Code: In a multiline SAS comment, proc or data step, other common SAS statements are checked to see if there is commented out code in the program. Appropriate text is written to the “Program Readability” column in the PI.

PROJECT WIND-UP ACTIVITIES:

13. CHECK-IN THE PROGRAMS, MAKE THE OUTPUTS READ-ONLY

The utility can perform the following tasks when appropriate configuration parameters are set:

- a) Check in the programs that are either checked out or not in version control system by running the specified OS commands for each program.
- b) Make the outputs read-only, by running appropriate chmod commands for each output.

14. REDUNDANT PROGRAMS/FILES IN PRODUCTION/VALIDATION LOCATION

The utility creates a list of programs/files that are present in the project directory and are not there in the PI. There is an option to skip particular files from this list, e.g. setup files. This list can be useful while cleaning up the project directories and is a list of files that are potentially not needed (created temporarily for testing purposes or backup copies etc.)

DESIGN CONSIDERATIONS AND IMPLEMENTATION

The utility is implemented for unix SAS 9.2, but can be ported to unix SAS 9.1.3 or PC SAS 9.1.3/9.2 with some limitations. Table 4 below describes the various SAS features needed for various platforms:

Feature	SAS 9.1.3		SAS 9.2	
	Unix	Windows	Unix	Windows
Reading/writing to single sheet MS Excel file	proc import /export	proc import /export	proc import /export	proc import /export
Reading/writing to multiple sheet MS Excel file	proc import + excelxp tagset	proc import /export	proc import /export	proc import /export
Fetching file modified date/time	Pipe option in filename statement	Pipe option in filename statement	Pipe option in filename statement/ FINFO function	Pipe option in filename statement/ FINFO function
Fetching file read-only attribute	Pipe option in filename statement/ FINFO function	Pipe option in filename statement	Pipe option in filename statement/ FINFO function	Pipe option in filename statement

Table 4. SAS features needed for implementation on various platforms

FUTURE DEVELOPMENTS

There are seemingly endless possibilities of attaching other “plug-in” utilities to this utility because Program Index is a central location of storing vital information about the project components.

“Program Readability” is quite a “qualitative” characteristic of a SAS program that I have tried to quantify. The sample SAS codes presented in this paper are not foolproof. I do intend to continue to work on this concept and further perfect the algorithms to quantify the indentation of other SAS blocks, e.g. if-then-else, do-end, select-when-end, %if-%then-%else and %do-%end.

CONCLUSION

Automation of various project activities can greatly enhance a team’s ability to timely and effectively deliver standardized outputs for a regulatory submission. The utility aims to achieve this goal and creates a programming environment where a lead programmer can automate some routine tasks effectively and accurately track the project status and the team can go to a central location and see what items are pending and need fixing.

FOOTNOTES

[#] these features are similar to the java application, PI Manager, developed by Tony Chang & Dana Soloff, Amgen Inc.
⁶ Please refer to reference number 6 below for implementation details of log checking.

REFERENCES

1. "A Simple Solution for Managing the Validation of SAS® Programs That Support Regulatory Submissions" by Tony Chang & Dana Soloff, Amgen Inc. http://www.sascommunity.org/mwiki/images/f/fb/Chang_and_soloff.pdf
2. SAS Online Documentation for version 9.1.3 and 9.2.
3. "Let your title macro report study progress" by Yang Chen, Consultant, Jersey City, NJ <http://www.pharmasug.org/cd/papers/AD/AD07.pdf>
4. "The 10 Most Frequently Asked Questions of Exporting to Excel®" http://www.mwsug.org/proceedings/2010/excel_db/MWSUG-2010-113.pdf
5. Excelxp tagset demo: http://support.sas.com/rnd/base/ods/odsmarkup/excelxp_demo.html
6. "You've Got E-Mail: Automatic Log Checking Via E-mail Notification", Aaron Augustine, Deloitte & Touche LLP, Prasenjit Dutta, Deloitte & Touche Audit Services India Private Limited. <http://www2.sas.com/proceedings/sugi31/128-31.pdf>
7. "An Introduction to Perl Regular Expressions in SAS 9" by Ron Cody, Robert Wood Johnson Medical School, Piscataway, NJ. <http://www2.sas.com/proceedings/sugi29/265-29.pdf>
8. GNU RCS: <http://www.gnu.org/software/rcs/rcs.html>

ACKNOWLEDGMENTS

I would like to thank Jack Fuller and Scott Davis for reviewing this paper and providing valuable comments.

CONTACT INFORMATION

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

Author Name: Rajesh Lal
Company: COMSYS - A Manpower Company
Address: 5220 Lovers Ln Ste 200
City state ZIP: Portage, MI - 49002
Work Phone: 269-553-5147
Fax: 269-553-5101
E-mail: rlal@comsys.com
Web: www.comsys.com

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.