PharmaSUG China 2018 – Paper AD-58

TLF Management Tools: SAS programs to help in managing large number of TLFs

Eduard Joseph Siquioco, PPD, Manila, Philippines

ABSTRACT

Managing countless Tables, Listings, Figures (TLFs) in a study is a very time-consuming task for new and veteran programmers. Manually checking the run times, validation results, log errors can be very tedious and tiring for the study lead. Using SAS directory functions with basic SAS functions, we can extract more information from our programs and use this to our advantage. This paper will explore the essential codes on the management tools and showcase the efficiency on the TLF management tools.

The TLF management tool will be able to identify TLF programs which have common issues such as run sequence and mismatches between development versus validation program, and log issues. SVN-managed environments will also have the option to display the owner of the programs. Aside from all this, macros used by different programs will also have a report to notify if the macro-dependent programs are run after any macro updates.

INTRODUCTION

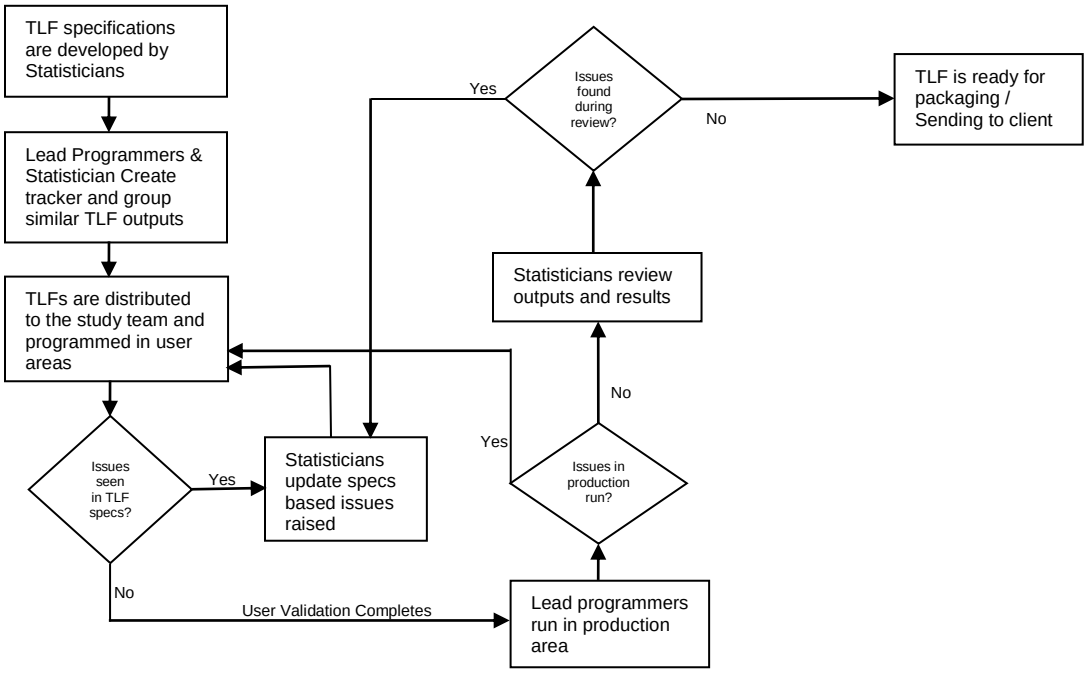
Individual TLF work might be a simple task for the team programmers but there is a lot of QC, documentation and managing work that must be done by the lead programmers. In short TLF lists, doing this manually would be the logical way to go since there won't be too much loss of time and it would be the easier option for the lead. In large TLF lists, doing this manually would lead to the massive loss of time and more stress incurred by the lead programmer.

The first idea that might come to mind would be creating some applications to provide a report or summary using other programming languages but in fact- SAS can also be used to achieve in creating the report or summary that the programmers need.

TLF WORKFLOW

In the creation of any report or summary tool it should be established what is the current process of the TLF development cycle. The diagram below will show the common TLF development cycle.

Flowchart 1. TLF development cycle



TLF MANAGEMENT TOOLS

This section will showcase three useful tools for TLF development work and explore the main code components of each.

Reading in the directory

To check for the status of the TLFs the tool will need to read and process the contents of our TLF directory. Most SAS programmers in the industry often forget that SAS has some powerful tools to help us achieve this task. We'll use the I/O functions that are readily available in SAS.

```
data files_all; *Create a dataset containg all the files in the current directory;
keep file_name ext;
length fref $8 file_name $80 ext $10;
        rc=filename(fref,".");
        if rc=0 then do;
                did = dopen(fref);
                rc = filename(fref);
        end;

        else do;
                msg = sysmsg();
                put     msg=;
                did =.;
        end;

        if did <= 0 then putlog 'Error: Unable to open directory.';
        filesnum= dnum(did);

        do i = 1 to filesnum;
                file_name = upcase(dread(did, i));
                fid = mopen(did, file_name);
                ext=scan(file_name,-1,'.');
                if fid > 0 then output;
        end;
        rc = dclose(did);
run;
```

The filename function contains the variable the directory of the files is assigned. In the code above FREF variable is assigned to the current directory of the where SAS is started. The wildcard location "." can also be changed to any location with the usual syntax "C:\Users\siquioej\Documents\Pharmasug" for example. DOPEN function is required so that other functions for the files members in the directory will work. DNUM function will return the number of members in the directory. DREAD function will return the *ith* file from the directory. MOPEN function is like DOPEN function but is used in files; in this code it is only used to validate that file exists. The output for the code above is shown below.

Output 1. Dataset output after reading files via SAS functions only

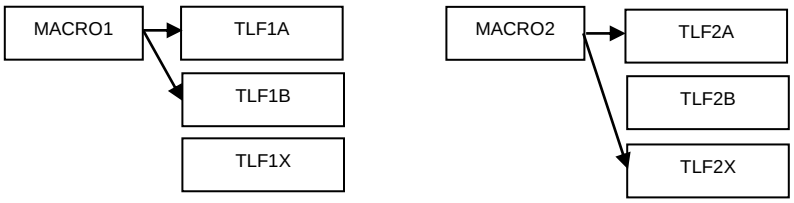
VIEWTABLE: Work.Files_all

	file_name	ext
1	F0801.LOG	LOG
2	F0801.SAS	SAS
3	F0801.SAS7BDAT	SAS7BDAT
4	F0801B.LOG	LOG
5	F0801B.SAS	SAS
6	F0801B.SAS7BDAT	SAS7BDAT
7	F0802.LOG	LOG
8	F0802.SAS	SAS
9	F0802.SAS7BDAT	SAS7BDAT
10	F0802B.LOG	LOG

This dataset will be then used as a macro input for the tools below. *File_name* will be made into a macro variable *file_current*. and looped thru all necessary files.

TOOL 1: MACRO LIST

TLFs with minor difference such as subset and variables changes will greatly benefit from using macros. The programmers assigned to these group of unique TLFs along with the repeats often create macros that will be very hard to track in a study. In this tool we are using the SAS programs themselves to give us the data information from the macros that they are using. Consider a scenario where a macro program called *MACRO1.sas* is being used for *TLF1A.sas* and *TLF1B.sas* and *MACRO2.sas* is being used for *TLF2A.sas* and *TLF2X.sas* – If we are a lead of the study we would manually check which programs will use this macro so we will know which outputs will be affected for every update of the unique TLF/macro program.



In this tool, the data step *infile* is being utilized. The tool will be using the sas programs itself to provide the data that the tool needs. From the dataset in the previous section, a code is implemented to create a macro variable *file_current* that is equivalent to the sas file names (TLF1a.sas, TLF1b.sas, etc.) and use a do loop for the input code.

Data input code

```
data current_prog(where=(index(upcase(line), "%INC")));*Include lines with %INC;
              length line $1000 file_current $80;
              infile "&file_current." dlm='$' missover;
              *May have to change delimiter if does not work right;
              input line $;
              file_current="&file_current.";
run;
```

The output of this dataset will be appended onto a final dataset which will be used to process the data obtained. Since different programmers have different styles in using the %include statement the program must process the code line as robust as possible. This can be done via prxmatch function or multiple tranwrd calls. The final dataset Macro_list may be used in a proc export statement in order to display the results in a more convenient format in which filters will be easier applied.

Output 2. Raw data into processed data

VIEWTABLE: Work.Append_prog

	line	file_current
1	%include "fmscatter.sas";	F0801.SAS
2	%include "fmscatterb.sas";	F0801B.SAS
3	%include "fmscatter.sas";	F0802.SAS
4	%include "fmscatterb.sas";	F0802B.SAS
5	%inc tmprotdev;	T0603B.SAS
6	%inc tmprotdev;	T0606.SAS
7	%include "tmae1.sas";	T0802.SAS
8	%include "tmae1.sas";	T0802B.SAS
9	%include "tmae2.sas";	T0803.SAS
10	%include "tmae2.sas";	T0803B.SAS
11	%include "tmae3.sas";	T0804.SAS
12	%include "tmae3.sas";	T0804B.SAS
13	%include "tmae1.sas";	T0806.SAS
14	%include "tmae1.sas";	T0806B.SAS

VIEWTABLE: Work.Macro_list

	program_name	macro
1	F0801.SAS	FMSCATTER.SAS
2	F0801B.SAS	FMSCATTERB.SAS
3	F0802.SAS	FMSCATTER.SAS
4	F0802B.SAS	FMSCATTERB.SAS
5	T0603B.SAS	TMPROTDEV.SAS
6	T0606.SAS	TMPROTDEV.SAS
7	T0802.SAS	TMAE1.SAS
8	T0802B.SAS	TMAE1.SAS
9	T0803.SAS	TMAE2.SAS
10	T0803B.SAS	TMAE2.SAS
11	T0804.SAS	TMAE3.SAS
12	T0804B.SAS	TMAE3.SAS
13	T0806.SAS	TMAE1.SAS
14	T0806B.SAS	TMAE1.SAS

Output 3. Final output of tool

	A	B
1	program_name	macro
2	F0801.SAS	FMSCATTER.SAS
3	F0801B.SAS	FMSCATTERB.SAS
4	F0802.SAS	FMSCATTER.SAS
5	F0802B.SAS	FMSCATTERB.SAS
6	T0603B.SAS	TMPROTDEV.SAS
7	T0606.SAS	TMPROTDEV.SAS
8	T0802.SAS	TMAE1.SAS
9	T0802B.SAS	TMAE1.SAS
10	T0803.SAS	TMAE2.SAS
11	T0803B.SAS	TMAE2.SAS
12	T0804.SAS	TMAE3.SAS
13	T0804B.SAS	TMAE3.SAS
14	T0806.SAS	TMAE1.SAS

TOOL 2: TLF RUN SUMMARY

After user area development the TLFs will be run in the production area and thus might encounter some unforeseen log issues or mismatches in validation. Checking the run status of each TLF would be a very tedious task for anyone managing the study. Using simple SAS steps along with the core component discussed earlier in the section, the task can be turned into a simple summary report.

Log check summary

Checking the log is an important part in assuring the quality of the TLF outputs. There should not be any errors, warnings, uninitialized variables, merge by issues, conversion issues and other log issues present. There are several papers available already for checking these. In this paper, we'll be exploring how we can use the reports of these outputs into one summary file with the validation.

Output 4. Output from a log check tool

```
*****************************

Search Completed (UTC): 2018-07-16 02:58:59Z
Files Searched:                          1
Number of REQUIRED but Missing Strings: 0
Number of PROHIBITED Strings:           25
Number of RESTRICTED Strings:           28
Number of CONDITIONAL Strings:          0
Number of INFORMATIONAL Strings:        0
```

Output 4 shows a sample of a output of a logcheck tool which search individual programs and outputs an additional file that shows the log issues and the summary. The tool is using this output and processing it to create a preliminary dataset for further processing.

Output 5. Preliminary output from a log check tool

VIEWTABLE: Work.Process2_chk

	file_current	chk_4	chk_3	chk_2	chk_1
1	F0801.CHK	Conditional			
2	F0801B.CHK	Conditional	Restricted		
3	F0802B.CHK		Restricted		
4	F0803B.CHK		Restricted		
5	F0804.CHK		Restricted	Prohibited	
6	F0804B.CHK		Restricted		
7	F0805B.CHK		Restricted	Prohibited	
8	VF0804.CHK		Restricted	Prohibited	
9	VF0805B.CHK		Restricted	Prohibited	

The dataset shown above contains all the information needed for the report but not yet as concise as we want it to be. If we notice observations 5 and 8 , it is shown that they are paired log check results. One is for the development side and one for the validation side. Adding some more post processing to the tool via subset and merges, the tool will come up with a dataset that would show the output name instead of the individual program names for the log check files. The final dataset we'll use to compile our data will be looking like the output below. Log issues are shown by columns per side of programming.

Output 6. Final dataset output of log check tool

VIEWTABLE: Work.Issues_chk

	deliverable_id	log_dev	log_val
1	F0801	Conditional	
2	F0801B	Conditional, Restricted	
3	F0802B	Restricted	
4	F0803B	Restricted	
5	F0804	Restricted, Prohibited	Restricted, Prohibited
6	F0804B	Restricted	
7	F0805B	Restricted, Prohibited	Restricted, Prohibited

Validation Summary

In TLF and other programming tasks, ensuring quality would mean parallel programming of one output by two programmers. There will be a development side and a validation side. The validation code would always end their program with a proc compare to check for any mismatches in the output.

The proc compare also produces a SYSINFO automatic variable that would represent the results for the proc compare output: 0 for perfect match and non-zero for the different mismatches ranging from attribute difference to data-level mismatches. If the programmers can add a bit of code at the end of their validation programs there will be a common dataset that can be used as an input for the validation summary.

Output 7. Sample output for a SYSINFO program

VIEWTABLE: Work.Tlf_compares

	deliverable_id	avu_sysinfo	avu_timestamp
1	F0801	0	05JUL2018:14:28:19
2	F0801B	0	05JUL2018:15:16:56
3	F0802	0	05JUL2018:14:28:26
4	F0802B	0	05JUL2018:15:17:01
5	F0803	0	05JUL2018:13:07:53
6	F0804	0	05JUL2018:13:08:04
7	F0805B	0	05JUL2018:13:08:14
8	F0803B	0	05JUL2018:13:07:59
9	F0804B	0	05JUL2018:13:08:09

The snippet above shows the following *deliverable_id* has matching results with the validation side. This is considered a validated TLF. All non-zero SYSINFO values are considered mismatches.

An extra layer can also be added using the. lst file for reading the output of the comparison results. A snippet below is from a proc compare outputs, one snippet is a fully validated file and the other snippet is for a mismatching validation result.

Output 8. Proc compare snippets

```
Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 161.

NOTE: No unequal values were found. All values compared are exactly equal.
```

```
Number of Observations with Some Compared Variables Unequal: 164.
Number of Observations with All Compared Variables Equal: 0.
```

These outputs of the validation side can be used to check on the mismatches via importing the .lst files. Using a similar input code like Tool 1 we can extract the filename of the validation deliverable id of the files with issues. The data subset will change to containing the “Unequal” string while remove the string with 0 unequal observations,

```
data current_lst(where=(index(upcase(line), "UNEQUAL")));
        length line $1000 file_current $80;
        infile "&file_current." dlm='$' missover;
        *May have to change delimiter if does not work right;
        input line $;
        file_current="&file_current.";
        *Keyword is unequal - Code below is to remove lines where unequal is present but result is equal;
        if line ne 'Number of Observations with Some Compared Variables Unequal: 0.';
        if line ne 'NOTE: No unequal values were found. All values compared are exactly equal.';
        if line ne 'Number of Variables Compared with Some Observations Unequal: 0.';
        if line ne 'Total Number of Values which Compare Unequal: 0.';
run;
```

SVN Authors

In SVN managed environments the author of the last commit can be extracted. This will inform the lead programmer to distribute previous programs to their previous owners for a faster time updating or fixing issues. The code below will do the

```
filename FILE pipe "svn list -v ""file:///&g_repo.""";

data FILES ;
  length line text $200.;
 infile File missover pad length=len;
   input @01 line $varying200. len eof;
   text=upcase(strip(line));
   keep text;
run;
```

Output 9. Dataset output of SVN list

VIEWTABLE: Work.Files

	text
1	8413 USERNAM1 1352 APR 20 05:01 F0801.SAS
2	8420 USERNAM1 1514 APR 20 06:27 F0801B.SAS
3	8403 USERNAM1 1605 APR 20 04:06 F0802.SAS
4	8420 USERNAM1 1638 APR 20 06:27 F0802B.SAS
5	8493 USERNAM2 10819 APR 20 09:25 F0803.SAS
6	9063 USERNAM1 5567 APR 25 02:59 F0803B.SAS
7	9293 USERNAM3 7874 APR 26 05:08 F0804.SAS
8	10541 USERNAM1 5124 MAY 09 01:55 F0804B.SAS
9	9293 USERNAM3 8481 APR 26 05:08 F0805B.SAS
10	8407 USERNAM4 1546 APR 20 04:37 VF0801.SAS
11	8417 USERNAM4 1586 APR 20 05:58 VF0801B.SAS
12	8407 USERNAM4 1567 APR 20 04:37 VF0802.SAS
13	8419 USERNAM4 1606 APR 20 06:23 VF0802B.SAS
14	8496 USERNAM4 1548 APR 20 09:41 VF0803.SAS
15	9061 USERNAM4 1608 APR 25 02:44 VF0803B.SAS
16	9198 USERNAM4 1553 APR 25 10:48 VF0804.SAS
17	9981 USERNAM4 1471 MAY 04 02:58 VF0804B.SAS
18	9198 USERNAM4 1578 APR 25 10:48 VF0805B.SAS

The pipe command tells as to communicate with another process, in our case this is the SVN list command for the console. It will input the command in the console and put the results in a sas dataset. The dataset output can be then processed to more manageable form and integrate in the program. The first set of numbers is the revision number and the username of the author will be separated with a space. The last string is the file name.

Output 10. Consolidated output for log summary and validation summary

	B	C	D	E	F	G	H	I
1	deliverable_id	avu_sysinfo	avu_timestamp	comments	log_dev	log_val	unequal_line	authors
2	F0801	0	7/16/2018		Conditional			usernam1/usernam4
3	F0801B	0	7/16/2018		Conditional, Restricted			usernam1/usernam4
4	F0802	0	7/16/2018					usernam1/usernam4
5	F0802B	0	7/16/2018		Restricted			usernam1/usernam4
6	F0803	0	7/16/2018					usernam2/usernam4
7	F0803B	0	7/16/2018		Restricted			usernam1/usernam4
8	F0804				Restricted, Prohibited	Restricted, Prohibited		usernam3/usernam4
9	F0804B	0	7/16/2018		Restricted			usernam1/usernam4
10	F0805B				Restricted, Prohibited	Restricted, Prohibited		usernam3/usernam4

After merging the results from the validation summary and log check summary the tool will export an excel file that would show easily the programs with validation issues and programs with compare issues. This report can easily be distributed to the team or let the programming lead know the overall health of the study.

TOOL 3: TIMESTAMP CHECKS

Rerunning the TLFs due to updates in specs/reviews and other issues will be inevitable and the leads must make sure that both development and validation side is always run. Also, when updating macros for an update in one TLF the other TLFs using the same macro should also be run to ensure consistency and quality of the outputs.

The main code component is using the same component as reading in the directory files, the program just needs to add one function FINFO and get the Last Modified date of the file.

```
do i = 1 to filesnum;
        file_name = upcase(dread(did, i));
        fid = mopen(did, file_name);
        ext=scan(file_name,-1,'.');
        if fid > 0 and upcase(ext) IN ('LOG' 'RTF' 'SAS' 'SAS7BDAT') then do;
                lastmod=finfo(fid, 'Last Modified');
                output;
        end;
end;
```

Output 11. Output dataset with the FINFO function for Last modified date

VIEWTABLE: Work.Files_all

	file_name	lastmod	ext
1	F0801.LOG	05Jul2018:14:27:56	LOG
2	F0801.SAS	05Jul2018:13:52:48	SAS
3	F0801.SAS7BDAT	05Jul2018:14:27:49	SAS7BDAT
4	F0801B.LOG	05Jul2018:15:16:30	LOG
5	F0801B.SAS	05Jul2018:15:12:53	SAS
6	F0801B.SAS7BDAT	05Jul2018:15:16:22	SAS7BDAT
7	F0802.LOG	05Jul2018:14:28:10	LOG
8	F0802.SAS	05Jul2018:13:52:48	SAS
9	F0802.SAS7BDAT	05Jul2018:14:28:03	SAS7BDAT
10	F0802B.LOG	05Jul2018:15:16:47	LOG

To make the output into a more readable format, the tool needs to add some processing to change the layout into a wider format. Processing the tool further would separate the validation side outputs from the development side ones and merge them back into one dataset for comparison and output. Using the dataset from the Macro tool, we'll match the logs of the programs to their respective macros if any.

Output 12. Merging time of Macro programs to individual programs

VIEWTABLE: Work.Timecomp_dev

	filesort	devtime	macroname	progtime
1	F0801.LOG	05JUL2018:14:27:56	FMSCATTER.SAS	05JUL2018:13:52:48
2	F0801B.LOG	05JUL2018:15:16:30	FMSCATTERB.SAS	25JUL2018:13:06:18
3	F0802.LOG	05JUL2018:14:28:10	FMSCATTER.SAS	05JUL2018:13:52:48
4	F0802B.LOG	05JUL2018:15:16:47	FMSCATTERB.SAS	25JUL2018:13:06:18
5	F0803.LOG	05JUL2018:13:03:56		.
6	F0803B.LOG	05JUL2018:13:06:30		.
7	F0804.LOG	05JUL2018:13:06:46		.
8	F0804B.LOG	05JUL2018:13:07:18		.
9	F0805B.LOG	05JUL2018:13:07:26		.

VIEWTABLE: Work.Timecomp_val

	filesort	valtime	valmacroname	valprogtime
1	F0801.LOG	05JUL2018:14:28:19	VFMSCATTER.SAS	25JUL2018:13:05:58
2	F0801B.LOG	05JUL2018:15:16:56	VFMSCATTER.SAS	25JUL2018:13:05:58
3	F0802.LOG	05JUL2018:14:28:26	VFMSCATTER.SAS	25JUL2018:13:05:58
4	F0802B.LOG	05JUL2018:15:17:01	VFMSCATTER.SAS	25JUL2018:13:05:58
5	F0803.LOG	05JUL2018:13:07:53	VFMPLOT.SAS	20APR2018:09:40:54
6	F0803B.LOG	05JUL2018:13:07:59	VFMPLOTVS.SAS	24APR2018:06:29:44
7	F0804.LOG	05JUL2018:13:08:04	VFMPLOTULN.SAS	26APR2018:04:59:00
8	F0804B.LOG	05JUL2018:13:08:09	VFMPLOTVS.SAS	24APR2018:06:29:44
9	F0805B.LOG	05JUL2018:13:08:14	VFMPLOTULN.SAS	26APR2018:04:59:00

The final output would be a excel file that contains the following rows. Comparing the timestamps of the macros, program logs to determine the sequence issues of the programs are listed in the first column of the program. This would help the leads to determine which programs to run and check for quality issues.

	A	B	C	D	E	F	G	H	I	J	K
1	Run_issue	filesort	Dev log Time	macroname	Macro Time	Val log Time	Val Macro time	valmacroname	rtfname	rtftime	Program List
2	Validation Macro updated, Validation not ran	F0801.SAS	7/5/2018	FMSCATTER.SAS	7/5/2018	7/5/2018	7/25/2018	VFMSCATTER.SAS	F0801.RTF	7/5/2018	VF0801.SAS
3	Macro updated, Development side not ran	F0801B.SAS	7/5/2018	FMSCATTERB.SAS	7/25/2018	7/5/2018	7/25/2018	VFMSCATTER.SAS	F0801B.RTF	7/5/2018	F0801B.SAS
4	Validation Macro updated, Validation not ran	F0802.SAS	7/5/2018	FMSCATTER.SAS	7/5/2018	7/5/2018	7/25/2018	VFMSCATTER.SAS	F0802.RTF	7/5/2018	VF0802.SAS
5	Macro updated, Development side not ran	F0802B.SAS	7/5/2018	FMSCATTERB.SAS	7/25/2018	7/5/2018	7/25/2018	VFMSCATTER.SAS	F0802B.RTF	7/5/2018	F0802B.SAS
6		F0803.SAS	7/5/2018			7/5/2018	4/20/2018	VFMPLOT.SAS	F0803.RTF	7/5/2018	
7		F0803B.SAS	7/5/2018			7/5/2018	4/24/2018	VFMPLOTVS.SAS	F0803B.RTF	7/5/2018	
8		F0804.SAS	7/5/2018			7/25/2018	4/26/2018	VFMPLOTULN.SAS	F0804.RTF	7/5/2018	
9		F0804B.SAS	7/5/2018			7/5/2018	4/24/2018	VFMPLOTVS.SAS	F0804B.RTF	7/5/2018	
10	Program sequence error	F0805B.SAS	7/25/2018			7/5/2018	4/26/2018	VFMPLOTULN.SAS	F0805B.RTF	7/5/2018	F0805B.SAS

CONCLUSION

Using tools for managing project or study work is a helpful way to alleviate any stress that comes in the manual and tedious checking of data files for quality. If these kinds of tools are deployed to a company the lead programmers and biostatisticians will have an easier time to obtain and digest all the information, giving them a clearer view on the progress and status of the TLF work in the study.

The creativity on the way we use SAS will often lead to similar outputs we can do by using other programming languages. SAS has a lot of functions for us to exhaust in creating these types of reports.

RECOMMENDED READING

- Reading and Processing the Contents of a Directory: https://www.mwsug.org/proceedings/2012/S1/MWSUG-2012-S128.pdf
- SAS log check program: http://support.sas.com/resources/papers/proceedings12/042-2012.pdf
- Summarize Multiple PROC COMPAREs Using the SYSINFO Macro Variable: https://www.lexjansen.com/pharmasug/2006/TechnicalTechniques/TT23.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Codes for the tools are available upon request. Contact the author at:

Name: Eduard Joseph Siquioco
Enterprise: PPD Inc.
Address: Manila, Philippines
E-mail: EduardJoseph.Siquioco@ppdi.com