PhUSE US Connect 2018

Paper PP09

Automating and Customizing TLG outputs with a Single Click

Bharathy Prasath, Zifo RnD Solutions, Chennai, India
Deepak Ananthan, Zifo RnD Solutions, Chennai, India

ABSTRACT

Most of the Clinical programmers would agree that the frustrating aspect about programming TLGs (Tables, Listings and Graphs) is to present the data in different file formats (pdf, rtf, xml etc.) and in different output types (all tables in the same file or separate files). The Statistician/Medical Writer usually wants all possible types of outputs and we mostly agree to it because we want to please them. The only time consuming thing here is writing repetitive codes for each type of output and running each code separately. This paper will provide an overview on simplifying this and lead you towards the process of automating most part of it.

INTRODUCTION

TLGs (Tables, Listings and Graphs) are usually generated as an input for the CSR (Clinical Study Report) and Safety Analysis Reports. Even though only the output as such is going to be used in the CSR, file formats and output types play an important role when the outputs are used for Safety/Interim Analysis, Sponsor review, Statistical review and Clinical interpretation. Both the Sponsor and the Statistician would want the outputs in a comfortable file format to ease their review. Moreover, each person has their own style. When all the Tables, Listings and Figures are put into the same file, it might be easy for the Medical writer to include this in the CSR, whereas, the Statistician would want to have the outputs in separate files so that they don't have to spend time loading or scrolling through thousands of pages. Some Statisticians prefer rtf outputs while others prefer pdf outputs. The sponsor's preference and point of view however is totally different. They would mostly prefer the outputs, especially Listings, as an excel file so that they can filter and sort what they need.

Considering all these preferences of the Sponsor, Statistician and Medical Writer, it is the Statistical Programmer's task to handle these conditions in his/her code. TLGs are all about the look, feel and interpretation. So, there is a great possibility of the sponsor changing their requirement in the eleventh hour. I think the Programmer would agree that even programming for the output type or format is easier compared to this change request from the sponsor to re-do it all again. Thus, predicting all the possibilities for changes and programming dynamically is really frustrating. It'd be a great relief if the Programmer had to look at only the actual Table, Listing or Graph and just let the automation do it for him/her.

This paper will give you the logic and some essential parts of the code behind this automation to make the sponsor feel at home even for a change request at the last moment. This paper will also give provide ideas on the further development of this automation for your sponsor specific customizations. There is also a mention of the type of input taken from the user and how this can be customized too. We will mainly be dealing with the format of the file, the grouping of the outputs within a file and generating a number of these grouped files.

The different types of file formats this paper will be covering are:

1. RTF (Rich Text Format)
2. PDF (Printer Definition File)

The different types of outputs this paper will be covering are:

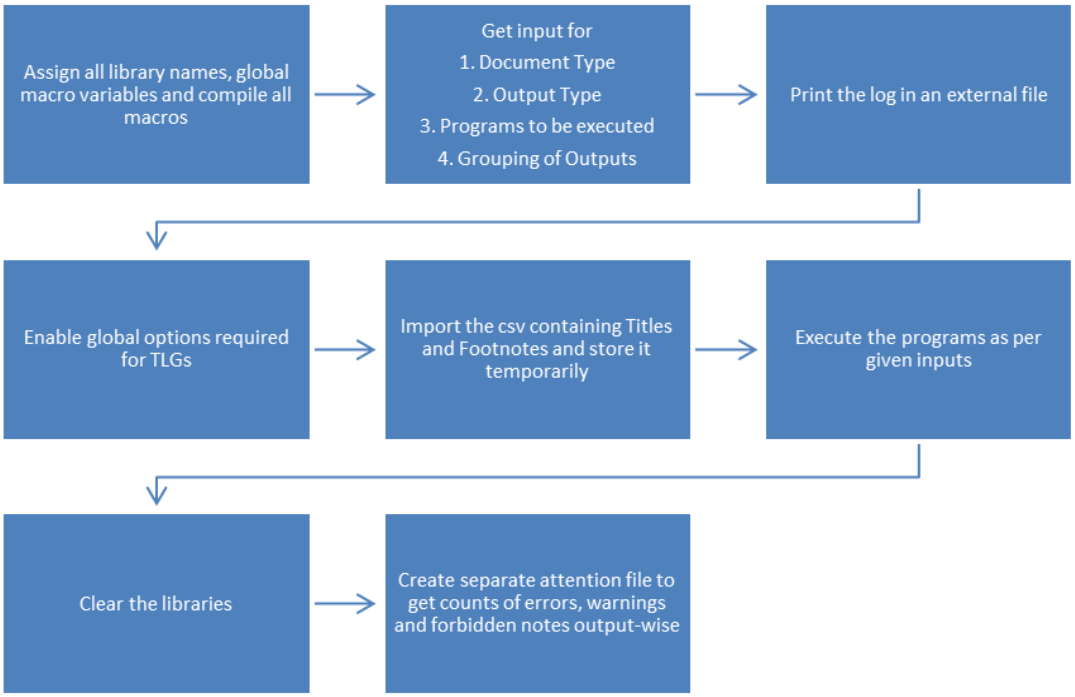
1. Consolidated output (All the outputs in a single file)
2. Individual output (Each output in a separate file)

We will also be covering the combination of the above. Though the formats mentioned above are limited, the logic explained in this paper can be extended to other file formats. Additionally, this paper will talk about accelerators that can be used for the creation of TLGs. The accelerators are:

1. Automation of Titles and Footnotes
2. TLG Report List
3. ErrWarnLog macro

FLOW OF TLG PROGRAMMING

Any clinical programmer would want as much automation as possible so that their work is made easier and they can concentrate on the results of an output and not the appearance aspect of it. Here, this point has been taken into consideration and the following is the flow of automation that we suggest. Our objective here is to minimize repetitive codes as much as possible. So, in the below suggested method there will be one program, say 00tlg.sas, which will execute all our TLG programs as per our inputs. Below is a flow chart depicting the flow of the 00tlg.sas program. We'll be discussing each step in the flow of 00tlg program in detail as we proceed further.



SETTING UP THE ENVIRONMENT

It is easier to set up the SAS environment completely and keep everything ready before proceeding to the actual coding. Otherwise, it'd be a little frustrating to keep making changes and going back to the initial stage and redoing everything from the beginning. For setting up the environment, we need to think of 3 things:

1. Libraries
2. Global macro variables
3. Compilation of macros

LIBRARIES

For ease of use, we assign different libraries for each purpose. For example, some of the datasets are created just intermediately and stored temporarily while some of them are used for QC purposes by the Validator. According to the purposes, libraries are assigned as follows:

```
x mkdir "&DIRPATH.Data\temp";
x mkdir "&DIRPATH.Outputs\TLG\TLG_&DATETIMESTAMP.";
libname dTEMP "&DIRPATH.Data\temp";
libname OTLG "&DIRPATH.Outputs\TLG\TLG_&DATETIMESTAMP.";
```

GLOBAL MACRO VARIABLES

One of the most time consuming work for the reviewer is to check consistency of repeating columns, headers etc. across all outputs. Some of the common values that are used across all programs can be assigned as global macro variables. This ensures that consistency is maintained throughout programs and developers. This also reduces the time spend for review on this. For example,

```
%global _STUDYID PROTOCOL SPONSOR AEMHDICT CMDICT LOGO DOCTYPE OUTPUTTYPE
DATETIMESTAMP;
```

```
%let _STUDYID = DEMO; /* Update the STUDYID to actual Study ID */
%let PROTOCOL = DEMO; /* The protocol number of the study */
%let SPONSOR = DEMO; /* The Sponsor name */
%let AEMHDICT=20; /*Adverse events/Medical History Coding Dictionary version number*/
%let CMDICT=WHODD; /* Concomitant Medication Coding Dictionary version number*/
%let LOGO=; /*Default is Zifo's logo. Update to customer specific logo if needed*/
```

COMPILATION OF MACROS

Opening each macro and compiling them before the execution of our program is a tedious job. Also, in this process, if we forget to compile some of the macros, it has to be compiled separately again. So, the compilation of macros is best done before starting with the programming. One easy way of automating this via SAS is:

```
* Include Validated Macros;
libname maclib "&DirPath.Programs\00 Macros\";
options mstored sasmstore=maclib;
* Include Study Specific Macros;
filename dirlist pipe %unquote(%bquote('dir /b ")&DirPath.Programs\00
Macros\%bquote(*.sas"'));
data _null_ ;
infile dirlist lrecl=200 truncover;
input line $200.;
if strip(left(line)) ne "sasmacr.sas7bcat" then
  call execute('%include "&DirPath.Programs\00 Macros\'||strip(left(line))||'" / lrecl
= 1000;');
run;
```

GETTING INPUTS

This part of the TLG automation takes input for the customization of the outputs as per the programmer's requirement hence satisfying the different requirements from the sponsor, safety reviewer, statistician etc. Similar to the previous section, this is also something, when done at the initial phase gives us more benefits than during programming. Inputs required for the TLG automation are:

1. Document Type
2. Output Type
3. Programs to be executed
4. Grouping of outputs

INPUT PROGRAMS AND GROUPING

During the final TLG preparation, there will be a lot of programs to be executed. Opening each one of them and executing them is a time consuming and tiring job. Instead, including all programs into a common program and executing just one program is an efficient alternative. This method also allows us to assign libraries, compile macros and assign global macro variables in one common place thus avoiding recurrences. An added plus on using this method is that, we can even analyze the SAS log for all programs together. The programs written by the developer are stored in a common location and are taken as input from the developer for execution and grouping through a CSV file as follows:

Program Name	Generate Report (Y/N/1/0)	Group
t-disposition	Y	tables
t-adverse events		tables
g-time trend plot	1	figures
g-km plot	Yes	figures

DOCUMENT TYPE

The document type is taken as input from the developer with an assign statement as below:

```
%let DOCTYPE = 0; /*0-consolidated document, 1-separate documents, 0 1-for both*/
```

When all the Tables, Listings and Figures are put into the same file, it might be easy for the Medical writer to include this in the CSR, whereas, the Statistician would want to have the outputs in separate files so that they don't have to spend time loading or scrolling through thousands of pages.

If DOCTYPE is assigned as 0, all the outputs are printed in the same rtf/pdf file according to the grouping given in the input CSV file. For example, from the table above, t-disposition and t-adverse events outputs will be printed in one rtf/pdf file for the "tables" group and the graphs for g-time trend plot and g-km plot will be printed in one rtf/pdf for the "figures" group.
If DOCTYPE is assigned as 1, outputs are printed separately. This does not take into consideration the "Group" column. When DOCTYPE was assigned as 0, the above table would give 2 outputs, one for each group. When it is assigned as 1, the table will give 4 outputs, one for each program.

OUTPUT TYPE

The output type is taken as input from the developer with an assign statement as below:

```
%let OUTPUTTYPE = RTF; /*RTF - RTF output, PDF - PDF output, RTF PDF - Both*/
```

If OUTPUTTYPE is given as rtf, outputs are printed in an rtf file. If OUTPUTTYPE is given as pdf, outputs are printed in a pdf file. If it is given as rtf pdf, both rtf and pdf outputs are generated. Since the ods statements for rtf and pdf differ in some options, these have to be handled differently. We'll be talking about the most efficient way to handle it later in the paper.

LOG AS AN EXTERNAL FILE

When programming the one thing that we keep forgetting to check time and again is the SAS log. Scrolling through each line of the SAS log for any Errors, Warnings or Forbidden Notes is both time consuming and frustrating. It'll be so much easier if we have the number of Errors, Warnings and Forbidden Notes ready and waiting for us once we run our program. The errwarnlog macro does the job of separating out the Errors, Warnings and Forbidden Notes from the long SAS log. It also gives us the count of each of these categories.

As an input for the macro, we store our SAS log externally. It is done as below:

```
filename TLGLOG "&LOGPATH.\TLGLog_&DATETIMESTAMP..log";
proc printto log = TLGLOG;
run;
```

ENABLING GLOBAL OPTIONS

Similar to most of the above sections, global SAS options are to be assigned in the beginning of the program. When two or more programmers work on a TLG project, it is likely that they follow different methods of representation for their outputs. If global SAS options are declared in common, then it'll bring the consistency that is required here. For example, below are some of the global options that might need consistency:

```
options orientation=landscape nonumber nobyline nodate center papersize=A4
ls=max ps=28 minoperator mindelimiter=",";
ods escapechar = "^" noresults;
```

TITLES AND FOOTNOTES

Assigning titles and footnotes for each output doesn't sound like much work. During the interpretation phase, due to discussions and clarifications, footnote changes are inevitable. Opening each program to make small changes is a lot of work. Not only for the programmer but also for the Validator since changing the program invalidates it. To avoid this, we can have one common file where all the Titles and Footnotes are present. The file can be in any format, for example, excel, csv etc. This file can be imported and used for each program. In case of a CSV file, it can be imported as follows:

```
proc import datafile="&DIRPATH.Programs\00 Config\TitlesFootnotes.csv"
out=DTEMP.TITLES_FOOTNOTES replace;
  getnames=yes;
  guessingrows=32767;
run;
```

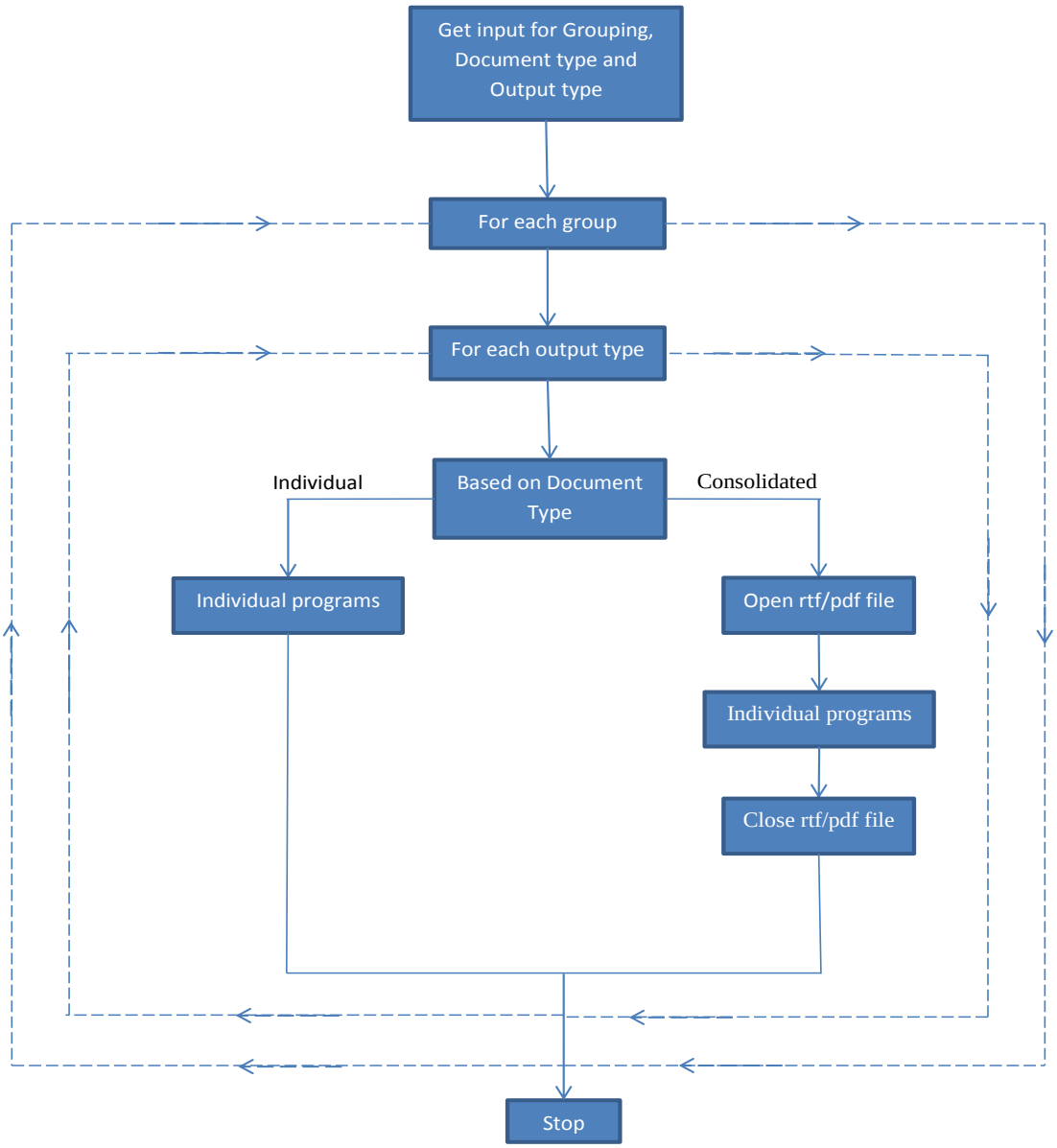
EXECUTING THE PROGRAMS

All the above steps can be done in one single program, say "the master program". The major reason for this being, execution of each program separately is a time consuming and tedious job. This also helps in automating the output type, document type and grouping of outputs.
The easiest and efficient process to incorporate all these options is to follow the below steps:

1. Depending on the required number of Groups and number of outputs in each group, individual programs can be combined and executed in the master program.
2. Within each Group, depending on the document type, programs can be executed in loops.

3. The whole process has to be repeated for each output type. This is not done along with the above methods since each output type requires different SAS features and options.

Sometimes, the sponsor requests for all the outputs to be combined into one file. This one file usually has thousands of pages and scrolling through them to check for accuracy of outputs is very much time consuming. To avoid this, we can split the outputs into logical groups. For example, all tables, all listings, all safety tables, all efficacy figures etc. Since the document type and output type require grouping first, this is taken as the first step in the master program. The number of groups can be counted and a loop can be run for each group. Inside this loop, the output type is taken into account next. Since the ods options vary between rtf and pdf files, the programs have to be run twice, once for rtf and again for pdf. Now, moving on to the document type, each document type has to be handled in a different way. If the outputs have to be printed in individual documents, the ods rtf/pdf statement, let's call it "odsstart", has to be called inside the individual programs. Else, if all outputs are required in the same file, then the odsstart has to be called once commonly for all the programs and not inside each program. After the loops are set up for the grouping, output type and the options are set up for document type, the individual programs are included into the master program and executed. For further clarity here is a flow chart depicting how the above steps are done:



The SAS program flow is as follows:

```
%do __DOC=1 %to %sysfunc(countw(&DOCTYPE.));
  %do __level = 1 %to %eval(&__GRPCNT.);
    %do __OUT=1 %to %sysfunc(countw(&OUTPUTTYPE.));
      *Open the ODS file;
      %if &__DOCTYPE.=0 %then %ODSSTART(FILE=&__STUDYID.-&__GROUP.-
      &DATETIMESTAMP.);;
      *Run for all active programs in a group;
      %include "&DIRPATH.Programs\03 TLG\&__RNAME..SAS" /lrecl=32767;
      *Close the ODS file;
      %if &__DOCTYPE.=0 %then %ODSCLOSE;;
    %end;
  %end;
%end;
```

BACK TO DEFAULT

It is always a good practice to change SAS back to its original self once our work with it is done. In the above process, we have changed several of the default SAS options and settings. We have assigned libraries, changed options and compiled macros. So, to return them back to original, the following SAS code is used:

```
%delWork;
options nomstored;
libname _all_ clear;
```

Apart from this, we have also been printing the SAS log outside in an external file. This is also not a part of SAS default settings. So, we return the printing of log back to SAS as below:

```
* Redirect External file to SAS log;
proc printto log=log;
run;
```

THE LOG SUMMARY

The automation for analysis of the externally printed SAS log does a great deal of time saving. A macro can be written to help us automate this. Giving out the counts for each category doesn't seem much complex. The real challenge here is to get parts of the SAS log that have issues. Internally, here in our organization we have a macro that does it all for us. The macro is used as follows:

```
%ERRWARNLOG(filname=&LOGPATH.\tlglog_&DATETIMESTAMP..log);
```

The output generated from the macro gives us an overall count of Errors, Warnings and Forbidden Notes in categories. It extracts parts of the SAS log that contain issues. For example, below is an output generated by the macro:

```
ERROR COUNT:7
WARNING COUNT:25
ATTENTION COUNT:29
NOTE ERR COUNT:22
UNINITIALIZED COUNT:0
TRUNCATED COUNT:0
MISSING VALUES COUNT:0
IMPROPERMERGE COUNT:0
GRAPH COUNT:0

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~GENERAL~~~~~~~~~~~~~~~~~~~~~~
NOTE: DATA statement used (Total process time):
real time           0.02 seconds
cpu time            0.00 seconds
WARNING: Multiple lengths were specified for the variable oid by input data set(s). This can
cause truncation of data.
NOTE: There were 14 observations read from the data set DSDTMQC.__DEFINE_VALUELEVELITEMREF.
NOTE: There were 234 observations read from the data set DSDTMQC.__DEFINE_ITEMREF.
NOTE: The data set DSDTMQC.__DEFINE_ITEMREF has 248 observations and 4 variables.
NOTE: Compressing data set DSDTMQC.__DEFINE_ITEMREF decreased size by 0.00 percent.

----------------------------------

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~t-disposition.sas~~~~~~~~~~~~~~~~~~~~~~
NOTE: PROCEDURE SQL used (Total process time):
real time           0.29 seconds
cpu time            0.04 seconds
Attention: DM domain Subject Reference Start Date/Time value is greater than the Subject Refe
rence End Date/Time value for this subject
_USUBJID=CLOBAL07771-B029-001
_RFSTDTC=
_RFENDTC=
NOTE: There were 449 observations read from the data set WORK.DM4.

----------------------------------
```

CONCLUSION

Programming Tables, Listings and Graphs can be time consuming and frustrating at times. Programmers tend to concentrate on the format, file types and technical aspects of the output rather than the data. Due to this, they tend to miss even an easy to catch data issue. The goal of the paper is to minimize technical work and automate most things for the programmers so that they can concentrate on the clinical data. Some of the automation that takes care of the above points is grouping, document type and output type. The automation of titles, footnotes, SAS log analysis, creating a master program etc. are accelerators that help us save time. Now that we have automated all of these, the question is for the future ahead. Most of the Validators would be tired of opening and checking the QC status document for every output. Not only that, but also, they would've had to switch between windows to review changes in TLGs for each Change Request. Our focus is now shifting from the Programmer to easing the lives of the Validators. What if you can get a list of changes from previous version in a single click? What if you can view the QC status of hundreds of outputs in a single click?

REFERENCES

RECOMMENDED READING

CONTACT INFORMATION

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

Author Name: Bharathy Prasath
Company: Zifo RnD Solutions
Address: 21A, Anna Salai, Littlemount, Saidapet, Chennai, Tamil Nadu 600015
City / Postcode: Chennai / 600015
Work Phone: +91 44 43114002
Fax:
Email: Bharathy.P@zifornd.com
Web: www.zifornd.com

Author Name: Deepak Ananthan
Company: Zifo RnD Solutions
Address: 21A, Anna Salai, Littlemount, Saidapet, Chennai, Tamil Nadu 600015
City / Postcode: Chennai / 600015
Work Phone: +91 44 43114002
Fax:
Email: Deepak.A@zifornd.com
Web: www.zifornd.com