PharmaSUG 2015 - Paper QT32

Accelerating Production of Safety TFLs in Bioequivalence and Early Phase

Denis Martineau, Algorithme Pharma, Laval, Quebec, Canada

ABSTRACT

This paper discusses the main steps that were made and how the SAS® software is utilized to maximize the output volume of Safety Tables, Figures, and Listings (TFLs) in bioequivalence and early phase studies using healthy volunteers. It explains how different teams within the same company sat down together to standardize their own study conduct processes in ways that facilitate each other's workflow. It also shows how small modifications to SAS code placed at strategic places make the creation of TFLs almost effortless in such environment.

INTRODUCTION

This paper originated about 6 years ago when a medium sized company envisioned how SAS could improve the quality of the reporting process and reduce the manpower required by the study reporting phase. After some quick math: 180 to 200 studies per year / 5 x 52 workdays - 10 holidays - 20 vacation and sick days = we estimated an 0.9 to 1 study report per 1.15 workday. That time includes reading the protocol and statistical analysis plan, extracting the data, running data checks, creating the TFLs, performing quality control of the TFLs, and finalizing the TFLs. This meant either hiring quite a few SAS Programmers to run such high volume of TFLs output or coming up with a solution to reduce the amount of programming time required for each stydy. The least expensive route was chosen obviously.

STUDY CONDUCT, A STANDARDIZED PROCESS

The first step is to meet with all groups involved, discuss how they work, how they have to work, and how they can adapt their method to accommodate other teams impacted by their work. The most notable adaptations came from the clinical team responsible for the raw data creation and the data management team responsible for the electronic data creation.

The clinical team adaptation included the creation of Case Report Form (CRF) templates to make sure the design was uniform from study to study in terms of data collected, page layout, visit naming; the identification of fields and pages seldom used; giving an electronic access to the unblinded planned randomization scheme which included blind treatment codes and actual treatment names; the use of a central lab capable of daily electronic transfer of results for all ongoing studies.

The data management team efforts included creating standardized entry screens based on the CDASH naming convention; granting the biostatistics team a read access to the Database Management System (DBMS); adding some informative variables to help customize the TFLs such as Sponsor ID and form version number which made it possible to update the clinic's CRF templates or customize TFLs to a sponsor's specifications.

AUTOMATING WITH SAS

Now that we have a standardized database design, we can start with the cool SAS programming.

Quick notes:

- Our Data Management team has two types of DBMS: a more traditional one called Clintrial and an Electronic Data Capture (EDC) system called Medrio. When applicable, I will identify for which system the program code is used. If not identified, the presented code is independent of DBMS.
- We use Enterprise Guide® software. Most macro variables used are prompts defined in the Enterprise Guide software template projects.

DATA EXTRACTION

First things first, we need data before we can create any TFL. As mentioned previously, we setup authorization levels so that we can connect to Clintrial's Oracle database using the Oracle libname's engine.

```
LIBNAME CT_Data ORACLE PATH="&instance"   /* Clintrial's instance (database) name */
                       SCHEMA="&protocol" /* Name of the study in Clintrial */
                       USER="&user"       /* Clintrial Username */
                       PASSWORD="&user_PASSWORD"  /* Clintrial Password */
                       ;
```

Once the libnames are successfully created and since we know that some data sets can be absent from the study design (Neurologic Evaluation for example), we can have SAS take a look at the available data sets and run the extraction manipulation only on those data sets. The idea is to look at the members' dictionary table and see which domain data sets are available for extract. Our data management has made it easier by beginning all of their data sets by the 2-letter domain name. The SAS code below shows how we do it for Clintrial.

```
%MACRO extract;

 PROC SQL noprint;

 CREATE table raw_ds as
 SELECT libname, memname, SUBSTR(memname,1 ,2) as domain
 FROM dictionary.members
 WHERE libname EQ "CT_Data"
/** WHERE Clause related to Clintrial **/
 AND   INDEX(memname, "_UPDATE") GT 1
 AND   INDEX(memname, "ENROLL")  EQ 0
 AND   INDEX(memname, "TAGS_")   EQ 0
 AND   INDEX(memname, "VCT_")    EQ 0
/** END OF WHERE Clause related to Clintrial **/
 ORDER by 3,2;

/** Count the number of data sets to extract **/
    SELECT count(*)
    INTO :nb_ds
    FROM raw_ds;

/** Identify the name of the data sets in the CT_Data library **/
    SELECT memname
    INTO :ds1-:ds99
    FROM raw_ds
    ORDER BY domain;

/** Identify the Domain name of the data sets for use in the RawData library **/
    SELECT domain
    INTO :domain1-:domain99
    FROM raw_ds
    ORDER BY domain;

QUIT;

/** Data Extraction **/
 %DO i=1 %TO &nb_ds;
    DATA rawdata.&&domain&i.;
     SET CT_Data.&&ds&i;
    RUN;
 %END;

%MEND extract;
%extract;
```

ANALYSIS DATA SETS

Before we proceed to the analysis data sets, please note that formats are nearly impossible to standardize. This is not a problem for our EDC because the formatted value and the coded value are both included in the extracted data sets. However in our Clintrial system we developed some code to read the different format names available in the DBMS and then identify which ones are assigned to which variables in a given study. Other DBMS such as Oracle Clinical also offer this possibility; all you need is a bit of knowledge about the system's Oracle tables. In our case, we created two Oracle libnames named after the schema: CTSCODES and CTSDD.

```
/* Read formats names and types */
DATA codelists;
 SET ctscodes.code_index;
 IF index(codelist,"$") EQ 0 AND aggregated EQ "YES";
 KEEP codelist codetype valuetype sasname;
RUN;

PROC SORT DATA=codelists;
 BY codelist;
RUN;

/* Read formats values and labels */
PROC SORT DATA=ctscodes.aggregated_codes OUT=codes(KEEP= codelist code value label
longlabel);
 BY codelist code;
RUN;

/* Create format input data set */
DATA library.ct_fmt;
 FORMAT label $200.;
 MERGE codelists (IN=in1) codes(IN=in2);
 BY codelist;
 IF in1;
 IF codetype EQ "FIXED" THEN type = "n";
 ELSE type = "c";
 DROP longlabel;
 RENAME code     = start
        codelist = fmtname;
RUN;

/* Run Proc Format on input data set */
PROC FORMAT LIBRARY=library CNTLIN=library.ct_fmt;
RUN;
```

After creating the formats catalog, we may now scan Data Management’s study design to determine which ones apply to which variables.

```
%LET study = study_name; /* Study name in Clintrial. A prompt can be used in   */
                         /* Enterprise Guide software                          */
DATA _null_;
 SET ctsdd.item; /* Clintrial’s version of SAS’ dictionary.columns view */
 IF protocol EQ upcase("&study");
/* Customization can be inserted here to remove unnecessary variables */
 IF codename NE " " THEN codename = TRIM(LEFT(codename)) || ".";
 ELSE delete;
 IF dtype EQ "TEXT" THEN codename = "$" || codename;
/* Now we create a macro variable named data set_variable containing the format name
*/
 mac_var = COMPRESS(SUBSTR(panel,1,2) || "_" || item_name);
 CALL SYMPUT(mac_var, codename);
 KEEP protocol panel item_name codename mac_var;
RUN;
```

Let’s say for example the Adverse Event (AE) data set has a serious/not serious variable named AESER on which a numeric format named SERIOUS should be used, the code above will automatically do the same as if we used the following.

```
%LET AE_AESER = SERIOUS.;
```

The advantage being that it creates a macro value for all formatted variables for the study and eliminates the risk of accidentally using a different format than the one used by Data Management.

To complete the analysis data sets section, we will now create the ADNE based on the sometimes present Neurologic Functions NE data set that also happens to sometimes contain a timepoint variable NETPT.

```
%MACRO adne;
/* If the NE data set exists NB_OBS is > 0 */
  PROC SQL noprint;
   SELECT count(*)
   INTO :nb_obs
   FROM dictionary.tables
   WHERE libname = "RAWDATA"
   AND memname = "NE";
  QUIT;

  %IF &nb_obs EQ 1 %THEN %DO; /* If NE data set exists then create ADNE */
/* If NETPT variable is used (NETPT_COUNT > 0), it will be added to ADNE */
 DATA netpt_count;
  SET rawdata.ne;
  IF netpt NE .;
 RUN;

  PROC SQL noprint;
   SELECT count(*)
   INTO :netpt_count
   FROM netpt_count;
  QUIT;

PROC SORT DATA= rawdata.ne OUT=ne;
 BY subjid visitnum pagekey pagerpt neseq;
RUN;

DATA mylib.adne;
 ATTRIB subjid      LABEL= "Subject ID"
        visitnum    LABEL= "Visit Number" length=$15 FORMAT=$visit.
%IF &netpt_count GT 0 %THEN %DO;
        netpt       LABEL= "Timepoint" FORMAT=6.2
        netptu      LABEL= "Timepoint Unit" FORMAT=&ne_netptu.
%END; /* End NETPT Handling */
        visrpt      LABEL= "Visit Repeat Number" FORMAT=2.
        neseq       LABEL= "Test Code"
        netest      LABEL= "Test Name"
        neres       LABEL= "Test Result" FORMAT=&ne_neres.
        nedesc      LABEL= "Describe if Abnormal";
 MERGE ne (IN=in1) mylib.adsl (IN=in2);
 BY subjid;
 IF in1 AND in2;
/** Insert Additional Data Handling Here. **/
RUN;
  %END; /* End &nb_obs EQ 1 Handling */
%MEND adne;
%adne;
```

"PUSH THE BUTTON" TABLES, FIGURES & LISTINGS

Since this paper is not about how to program the outputs, let's suppose there is an existing programs base that can be adapted using some of the tips that follow:

1. The most obvious is to copy the same check that triggers the analysis data set creation for optional data sets to trigger the related TFLs output.
2. Following the same line of thinking, there may be TFLs that need to be outputted only when some conditions are met. An example is a "Change from Baseline" summary which requires post-baseline values to be collected.

```
%MACRO TFL;

PROC SQL noprint;
 SELECT count(*)
 INTO :ucandoit
 FROM mylib.advs
 WHERE visitnum GT "000";
QUIT;


%IF &ucandoit GT 0 %THEN %DO;
/* INSERT PROGRAM HERE */
%END; /* End ucandoit */
%MEND TFL;
```

3. Now, let's say we have some sponsors requesting different data in the TFLs. An example of that would be to have 3 sponsors (003, 005 and 008) ask for the MedDRA Preferred Term (PT) whereas all others are content with the Lowest Level Term (LLT). We also use a Form version variable since data collection can be changed over time independently from sponsors. Here is how we use the sponsor ID variable, cleverly named SPONSOR, and the CRF page version variable, named PAGEV.

```
%MACRO Output;

DATA adae;
 SET mylib.adae;
 IF sponsor in ("003", "005", "008") THEN do;
   llt_term = pt_term; /* Makes it easier to report the same variable name. */
   CALL SYMPUT("pt_spons", "1");
 END;
 ELSE CALL SYMPUT("pt_spons", "0");
 IF pagev EQ "AE V1" THEN CALL SYMPUT("pagev", "1");
 ELSE IF pagev EQ "AE V2" THEN CALL SYMPUT("pagev", "2");
 ELSE IF pagev EQ "AE V3" THEN CALL SYMPUT("pagev", "3");
RUN;

/* Skipping ahead to the proc report at the end. */
/* For clarity, we will only report SUBJID, LLT_TERM, AESTDAT, and AEENDDAT */

PROC REPORT DATA=ae_list;
 COLUMN subjid llt_term aestdat
        %IF &pagev EQ 2 %THEN %DO; aeendat %END; /* End date appears on V2 only */
 ;
 DEFINE subjid    / "Subject ID" ORDER ORDER=internal
                   style(column)=[cellwidth=1.0in just=left]
                   style(header)=[just=left];
%IF &pt_spons EQ 0 %THEN %DO; /* Identifies a sponsor preferring LLT Term */
 DEFINE llt_term  / "Lowest Level Term"
                   style(column)=[cellwidth=3.0in just=left]
                   style(header)=[just=left];
%END;
%ELSE %IF &pt_spons EQ 1 %DO; /* Identifies a sponsor preferring PT Term */
 DEFINE llt_term  / "Preferred Term"
                   style(column)=[cellwidth=3.0in just=left]
                   style(header)=[just=left];
%END; /* There is room for more sponsor customization (create values &pt_spons > 1) */
 DEFINE aestdat   / "Onset Date"
                   style(column)=[cellwidth=1.0in just=left]
                   style(header)=[just=left];
%IF &pagev EQ 2 %THEN %DO; /* End date appears on V2 only */
   DEFINE aeendat   / "Resolution Date"
                     style(column)=[cellwidth=1.0in just=left]
                     style(header)=[just=left];
%END;
RUN;
```

4. Another recurring issue is fixing the automated process in case of an occasional empty output data set. It usually ranges from a listing of abnormal laboratory values when there are none to a study with no AE to report. Since we do not like debugging error messages that can be easily avoided, we created a simple macro that prevents the REPORT procedure from failing and allows us to write a nice message in the output TFL.

```
%MACRO empty_ds(ds_name);

%GLOBAL nb_obs;
PROC SQL noprint;
 SELECT count(*)
 INTO :nb_obs
 FROM &ds_name;
QUIT;

%IF &nb_obs EQ 0 %THEN %DO;

 DATA temp;
  page = 1;
 RUN;

 DATA &ds_name;
  SET temp &ds_name;
 RUN;

%END;

%MEND empty_ds;

/** Let's use it in the AE Listing just before the Proc Report and in its options **/

%empty_ds(ae_list);
PROC REPORT DATA=ae_list;
 COLUMN subjid llt_term aestdat
        %IF &pagev EQ 2 %THEN %DO; aeendat %END; /* End date appears on V2 only */
 ;
/* Skipping the DEFINE section */
%IF &nb_obs EQ 0 %THEN %DO; /* Global variable from the empty_ds macro. 0=No data. */
  COMPUTE after subjid;
    line "No subject was found for this listing.";
  ENDCOMP;
%END;
RUN;
```

5. Finally as mentioned at the beginning, we have access to the randomization scheme of the study. This is most useful to determine the number of treatment columns in the summary tables and to copy the treatment names entered by the medication team. Here is how we use this data to output an AE frequency by LLT term (less how we calculate the frequencies).

```
/** The following is usually included with the libnames execution **/
%GLOBAL ntreat trt1 trt2 trt3 trt4 trt5 trtcode1 trtcode2 trtcode3 trtcode4 trtcode5;
PROC SQL;
 SELECT count(distinct extrt)
 INTO :ntreat
 FROM  rando;

 SELECT distinct extrt, propcase(label), treatmentcode
 INTO :trt1-:trt99, :trtcode1-:trtcode99, :rando1-:rando99
 FROM  rando
 ORDER BY treatmentcode; /* Order of appearance is predefined in the data set */
QUIT;
```

```
/** Now a macro that creates a treatment format **/
%MACRO unblind;
PROC FORMAT LIBRARY=mylib;
 VALUE $extrt "&trt1" = "&trtcode1"
              "&trt2" = "&trtcode2"
%IF &ntreat GE 3 %THEN %DO;
              "&trt3" = "&trtcode3"
%END;
%IF &ntreat GE 4 %THEN %DO;
              "&trt4" = "&trtcode4"
%END;
%IF &ntreat GE 5 %THEN %DO;
              "&trt5" = "&trtcode5"
%END;
 ;
RUN;
%MEND unblind;
%unblind;

%MACRO Output;

/** We assign a column number based on the treatment variable EXTRT **/
DATA adae;
 SET mylib.adae;
  IF extrt EQ "&trt1" THEN DO;
    column = 1;
    OUTPUT;
  END;
  ELSE IF extrt EQ "&trt2" THEN DO;
    column = 2;
    OUTPUT;
  END;
  IF extrt EQ "&trt3" THEN DO;
    column = 3;
    OUTPUT;
  END;
/** ... **/
RUN;

/** A basic column header for the report **/
DATA _null_
  CALL SYMPUT("header1",PUT("&trt1", $extrt.));
  CALL SYMPUT("header2",PUT("&trt2", $extrt.));
  CALL SYMPUT("header3",PUT("&trt3", $extrt.));
  CALL SYMPUT("header4",PUT("&trt4", $extrt.));
  CALL SYMPUT("header5",PUT("&trt5", $extrt.));
RUN;

/** The Summary data set outputs results in variables COLUMN1-COLUMN&ntreat  **/
/** where &ntreat is the number of treatments calculated above.             **/
/** Now, let's print the output.                                            **/

PROC REPORT DATA=report nowd split='~' missing;
 COLUMN order llt_term column1-column&ntreat;
 DEFINE order        / " " ORDER ORDER=internal noprint;

%IF &nbr_report_column EQ 2 %THEN %DO;
 DEFINE llt_term     / " ~Parameter" style(column)=[cellwidth=7.4in just=left]
                       style(header)=[just=left];
%END;
```

```
%ELSE %IF &nbr_report_column EQ 3 %THEN %DO;
 DEFINE llt_term     / " ~Parameter" style(column)=[cellwidth=6.25in just=left]
                       style(header)=[just=left];
%END;
%ELSE %IF &nbr_report_column EQ 4 %THEN %DO;
 DEFINE llt_term     / " ~Parameter" style(column)=[cellwidth=5.1in just=left]
                       style(header)=[just=left];
%END;

/** You may add more number of treatment possibilities as per your needs. **/
 %DO j = 1 %TO &ntreat;
  DEFINE column&j / " ~&&header&j.~ " style(column)=[cellwidth=1.15in just=left]
                    style(header)=[just=left];
 %END;
RUN;
%MEND Output;
```

CONCLUSION

In conclusion, there are numerous ways SAS can be used to minimize the time spent programming TFLs. This paper covers only the ones I feel offer the best starting basis. Hopefully this paper gives you ideas and you will develop techniques that suit your working environment. Please remember to put in the effort needed to develop a plan of possible improvements; evaluate the time needed to implement each one; and think how often you will save time for each project. This will help you decide where to begin.

ACKNOWLEDGMENTS

A huge thank you is to all past and present members of the Biometrics team at Algorithme Pharma. Without them none of this would have been possible.

RECOMMENDED READING <HEADING 1>

SAS(R) 9.2 SQL Procedure User's Guide

CONTACT INFORMATION

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

Name: Denis Martineau
Enterprise: Algorithme Pharma
Address: 575 boul. Armand-Frappier
City, State, Zip: Laval, Qc, Canada, H7V 4B3
Work Phone: (450) 973-3155 ext. 2220
Fax: (450) 973-7866
E-mail: demartineau@algopharm.com