PharmaSUG 2018 - Paper AD-32

Automation of STDM dataset integration and ADaM dataset formation

William Wei, Merck & Co, Inc., Upper Gwynedd, PA, USA

Rinki Jajoo, Merck & Co, Inc., Rahway, NJ, USA

Susan Kramlik, Merck & Co, Inc., Upper Gwynedd, PA, USA

ABSTRACT

Ongoing Aggregate Safety Evaluation (OASE) is a key business imperative in enabling the clinical safety model. Integrated data specification and derivation are key parts of the necessary data pooling strategy. Currently, data integration at the compound level faces many challenges. Due to a drug's development history, variable attributes, such as name, type, and length, frequently are inconsistent across studies. Thus, when datasets are pooled for an aggregate analysis, programmers need to check the datasets and make conversions manually so that the attributes are consistent. Differences in variable name and type can cause inaccuracies, such as missing values, and differences in length can potentially cause truncation of the variable value if it is not checked thoroughly. These tasks are very time consuming and tedious, and there is the possibility of human error. The key to enable the OASE is to efficiently integrate the trial-level data to compound level in an automated fashion. The tool was developed to achieve the automation based on CDISC standard SDTM/ADaM guidance and to standardize the pooling datasets from different studies in order to produce protocol-independent variables. For example, ADaM datasets such as ADAE/ADLB/ADVS can be generated automatically using this approach and hence high-quality work can be delivered while reducing the time and resources.

INTRODUCTION

Data integration is a process that combines data from multiple sources in a consistent and traceable manner. It involves data pooling, standardization, and harmonization. The benefits of data integration include the following.

- It provides one standardized data source for all integrated analyses, including regulatory submissions, answering customer questions, data mining and enabling data transparency.
- It facilitates quick and accurate decision making utilizing data from across a compound.
- It allows data mining of all compound data, which can reduce the number of additional studies needed to answer clinical questions, resulting in lower compound development costs.
- It simplifies integrated analysis by utilizing the integrated database (IDB) as a single source of input data.

Data integration is a challenging, tedious, and time-consuming process. With the application of CDISC standards, a modularized programming approach can do the heavy lifting for us. Thus, automated ADaM dataset creation has become possible except for the ADSL dataset, which needs more protocol-level manipulation.

ADaM data integration typically requires two major steps:

1. A data pooling process, which involves pooling data from multiple sources.
2. Data harmonization is the process of comparing different variable definitions, attributes and values and combining them in a consistent manner that maintains data quality and accuracy.

Data pooling allows analyses of the compiled dataset. Therefore, pooled data represent a subset of the integrated information to be presented. Harmonization can be achieved through creating new variables, recalculating existing variables in a common way, or recording events and medications into a common

version of a dictionary. This process can happen before data pooling or after data pooling, and it depends on the sources and the history of data.

Here we introduce our tools to facilitate data integration and ADaM dataset automation design by utilizing common processing methods, approaches, and algorithms that are standardized as common modules. In combining these modules, many data integration steps can also be automated. The tools add value by reducing the chance of human error, reducing resource needs, and reducing tedious and frustrating work, which can improve employee satisfaction. The data pooling toolkit introduced in this paper is used to pool dataset together. The ADaM automation modules use object-oriented programming concepts and build independent functional macros using SAS®. This approach makes ADaM dataset automation feasible.

DATA POOLING TOOLKIT

The inevitable data pooling required by ADaM data integration depends on the data sources and protocol designs, as well the analysis requirements. Pooling can be applied to SDTM or ADaM datasets (Figure 1**Error! Reference source not found.**). For example, if SDTM data sources came from different versions across a long history, pooling SDTM data may induce a huge amount of data harmonization after the pooling. In that case ADaM dataset pooling may be a favorable approach for the individual study team's ADaM generation efforts. Otherwise, if SDTM dataset versions are similar and come from the same system, then SDTM dataset pooling may be a good start.

Figure 1. Two ADaM data integration process. 1. Pooling SDTM/SDTMPLUS dataset and then generating an integrated ADaM (iADaM) dataset, including other harmonization manipulations. 2. Pooling individual ADaM dataset and then process the harmonization.

No matter whether pooling using the SDTM or ADaM datasets, the procedures and issues the programmer face are similar. Three major obstacles encountered when pooling datasets are as follows

1. Variable truncation - inconsistent variable length across multiple studies.
2. Missing values- inconsistent variable type in different data sources.
3. Appropriate format use to convert values when needed.

POOLING TOOLKIT DESIGN

The pool toolkit is designed using a building block approach. It consists of two major modules (Figure 2): 1. Information collection module (%PooldsAttr). 2. Datasets stacking module (%PooldsAct).

The first module %PooldsAttr collects the following information and passes it to the next block:

- Collect the list of datasets and corresponding number of observations for each study.
- Gather the list of variables with differing types across studies and their assigned formats
- Extract the maximum length for each variable across all of the studies
- Synthesize information from above two bullets to create macro variables
- Output studies/datasets/obs/variable type to excel for user to review

The second module %PooldsAct is used to stack the datasets together from multiple data sources. It has the following functional blocks:

- Use variable length information from %PooldsAttr to create dummy datasets
- Change variable type to character using assigned format if the variable has inconsistent type
- Stack datasets to dummy datasets from different studies specified by user
- Convert user specified character variable to numeric variable using specified format
- Optimize the character variable length to the maximum data length in pooled datasets

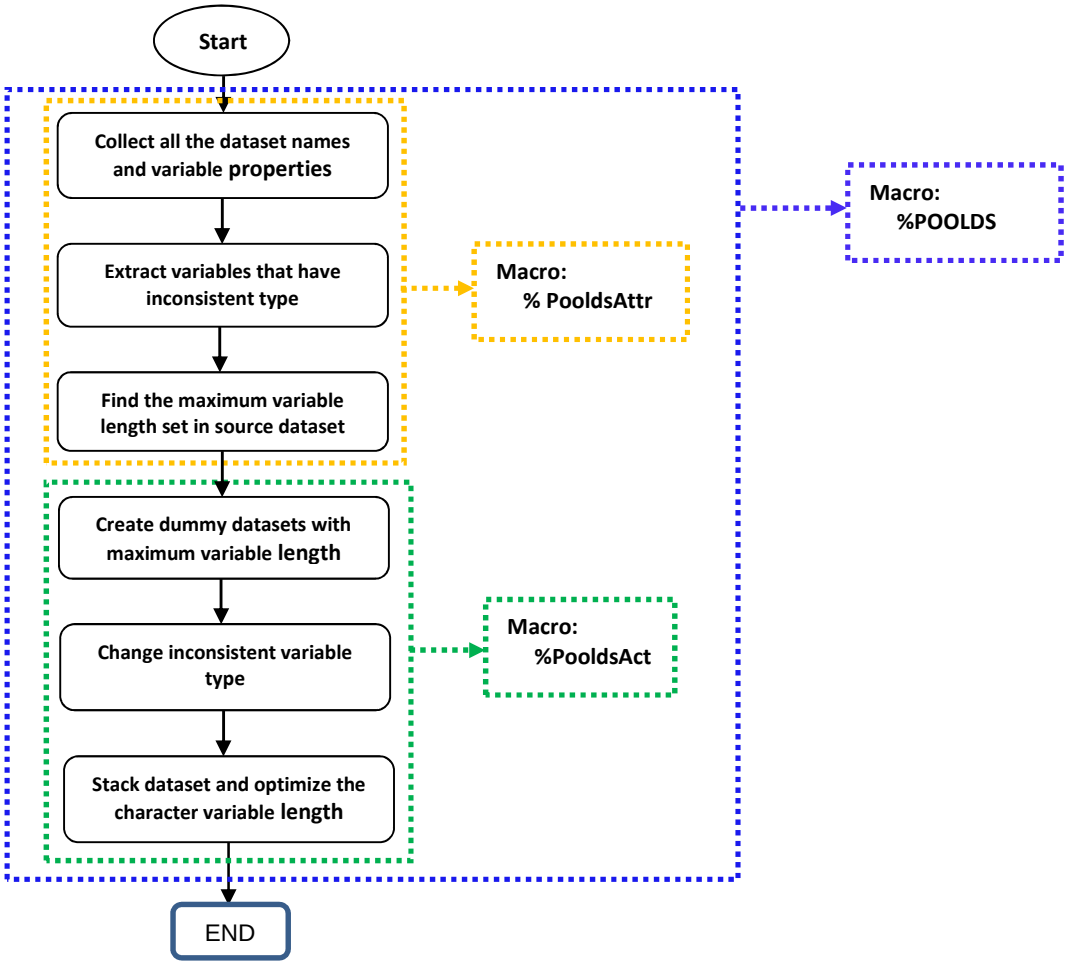

Figure 2. The flow chart of macro %poolds.

%POOLDSATTR MACRO MODULE

The main purpose of this module is to collect all information needed to prepare for dataset stacking. Using PROC DATASETS, all information needed could be collected in a combined dataset. In the following code, the SAS macro %splitter is commonly used to split a sentence to words. Here we used it to split the study list.

```
%* SPLIT STUDY LIST TO TOTAL NUMBER OF STUDIES: &STDTOT;
%* STUDY LISTS: STD1-STDN;
%SPLITTER(&STUDIES., WORDPFX=STD, DLM=%STR( ));

%* CREATE AN EMPTY DATASET ;
DATA _ATTR; RUN;
```

```
%DO I_STD = 1 %TO &STDTOT.;

      PROC DATASETS LIBRARY=&&STD&I_STD. MEMTYPE=DATA NOPRINT;
            CONTENTS DATA=_ALL_ OUT=WORK.ATTR_&&STD&I_STD.;
      QUIT;

      %* COMBINE STTRBUTE DATASET ;
      DATA _ATTR;
            SET _ATTR WORK.ATTR_&&STD&I_STD.;
      RUN;

%END;
```

The dataset _ATTR will have all the information we need. Then we are going to extract the information from _ATTR and transfer the information to the next module. Two major types of information are collected:

a. Variable/dataset list where variable type needs to be changed before stacking
b. Variable maximum length set in source dataset in different studies

This information can be collected using proc transpose. For example, to get the maximum length of each variable:

```
PROC SORT DATA=_ATTR OUT=_VARLEN NODUPKEY;
          BY MEMNAME NAME LIBNAME LENGTH; RUN;
PROC TRANSPOSE DATA=_VARLEN OUT=_VARLEN_T;
    BY MEMNAME NAME;
    ID LIBNAME;
    VAR LENGTH;
RUN;

%* GENERATE STUDIES LIBNAME LIST SEPERATED BY ',' ;
%LET STUBYCOM = %SYSFUNC(TRANWRD(%SYSFUNC(COMPBL(&STUDIES.)), %STR( ),
                                 %STR(,)));
DATA _LEN_MAX;
    SET _VARLEN_T;
    BY MEMNAME NAME;
    VARLENGTH= MAX(&STUBYCOM.);
RUN;
```

The next puzzle is how the information should be organized, easily passed to the other module and in a way, which keeps the module more independent for easy maintenance. There are many ways to accomplish this, for example, creating dummy datasets, storing information in a dataset, or storing information in macro variables. After comparing these approaches, we choose using macro variables to convey the information to the next module because of its' flexibility. The following are examples of macro variables to transport the information:

- The domain list which will be pooled:

  ```
  %let existdomain = domain1 domain2 domain3 …;
  ```

- Variables that type need to be converted:

  ```
  %let typchlst =
  LPTSS10.CF(ATTCKNMB:BEST,CFAEMHSQ:BEST,EXIVDOSE:BEST,MONTHNMB:BEST,STML
  TNDY:BEST,WEEKNMB:BEST)|LPTSS10.EX(EXDURDD:BEST)|LPTSS12.CF(ATTCKNMB:BE
  ST,CFAEMHSQ:BEST,CFKITID:BEST,EXIVDOSE:BEST,MONTHNMB:BEST,STMLTNDY:BEST
  ,WEEKNMB:BEST)|LPTSS12.EX(EXDURDD:BEST)| … ;
  ```

Note: The value of this macro variable format is repeat of unit “libname.dataset(variable1:format1, variable2:format2, …)” which is separated by “|”. Libname presents study folder. All the numeric variables whose types are inconsistent across studies are list in this macro variable.

- Maximum variable length from source datasets:

```
%let ae_var_len = STUDYID $8 DOMAIN $2 USUBJID $19 AESEQ 8 AEGRPID $2
            AEREFID $1 AESPID $3 AETERM $125 AEMODIFY $1 AEDECOD $62
            AECAT $1 AESCAT $15 AEOCCUR $20 AEBODSYS $80 AELOC ...;
%let cm_var_len = STUDYID $8 DOMAIN $2 USUBJID $19 CMSEQ 8 CMGRPID $5
            CMREFID $1 CMSPID $3 CMTRT $200 CMMODIFY $1 CMDECOD $255
            CMCAT $35 CMSCAT $39 CMOCCUR $20 CMSTAT $8 ...;
```

… …

Note: Macro variable name started with dataset name. The value of the variable could be used directly to generate dummy dataset by “length” statement.

%POOLDSACT MACRO MODULE

After the information has been collected by %PooldsAttr, the dataset pooling process is ready. First, we created dummy datasets for each pooling dataset. The maximum variable length set macro variable (XX_var_len) was used to build dummy dataset using following code:

```
%* SPLIT DOMAIN LIST TO TOTAL NUMBER OF STUDIES: &DOMTOT;
%*        SPLIT LISTS: DOM1-DOMN;
%SPLITTER(&DOMAINS., WORDPFX=DOM, DLM=%STR( ));

%DO I_DOM = 1 %TO &DOMTOT.;
  %* CREATE DATASET TEMPLATE ;
  DATA &&DOM&I_DOM.._POOL_MAXLEN;
    LENGTH &&&&&&DOM&I_DOM.._VAR_LEN.;
        CALL MISSING(&&&&&&DOM&I_DOM.._VAR_LST.);
        DELFLAG = 'Y';
        LABEL DELFLAG = 'Record delete flag';
  RUN;
%END;
```

Note: The content of macro variable XX_var_lst is similar to with macro variable XX_var_len if variable length were to be removed.

When a dummy dataset is created, then we need to perform a small surgery on the dataset before it can be stacked to the dummy dataset. If a variable has different types in different studies, we need to unify the variable type to make it possible to stack to the dummy dataset. To keep as much information as possible, we change the numeric type to its corresponding character type. For example, if a variable is numeric type in one study, but the same variable is character type in the other study, then we convert the numeric version to character version using their assigned format. Conversely, if we convert character variable to numeric, there are possibilities that we lose some information that cannot be recovered. For example, when we convert a partial date to its’ numeric version, it would irreversibly be changed to a missing value.

After stacking the datasets from all required studies, the macro provides an option to convert a character variable to numeric as user specified.

The final step of this module is to find the optimal length of character variables and assign the optimal length to the variable to be CDISC compliant. The sub-macro %optlen performs this function (the following code). In the following code, the first SQL procedure produces a one-record dataset that has the same number of variables as the input dataset. But the value of each input dataset variable is “variable

length=XX" for a character variable, and "variable" for a numeric variable. Data _NULL_ is used to combine all of the values together separated by ',' and assign it to a macro variable &_VARATTIB. The second SQL procedure uses this macro variable &_VARATTIB to set the length for the input dataset.

```
%* GET THE NUMBER OF VARIABLES ;
%LET DSID = %SYSFUNC(OPEN(&DsIn.));
%LET TVARNUM = %SYSFUNC(attrn(&DSID.,NVARS));

%* get the number of objects ;
%LET OBJNUM = %SYSFUNC(ATTRN(&DSID,NOBS));;

%* GET THE MAXIMUM LENGTH OF CHARACTER VARIABLES;
%IF "&OBJNUM." = "-1" %THEN %DO;
  %PUT &DsIn. READING ERROR, PLEASE CHECK IF IT EXIST OR CORRUPTED!;
  %LET DSCLOSE = %SYSFUNC(CLOSE(&DSID.));
  %GOTO MABORT;   %* ABORT ;
%END;
%ELSE %DO;
  PROC SQL NOPRINT;
  CREATE TABLE _OPTLEN AS
  SELECT
        %DO VNUM=1 %TO &TVARNUM.;
              %IF %SYSFUNC(VARTYPE(&DSID,&VNUM.))=C %THEN %DO;
                    CAT("%SYSFUNC(VARNAME(&DSID,&VNUM.))",
                          ' ',
                          'LENGTH=',
              IFN(MAX(LENGTH(%SYSFUNC(VARNAME(&DSID,&VNUM.)))) <= 1, 1,
                    MAX(LENGTH(%SYSFUNC(VARNAME(&DSID,&VNUM.)))) )
                          ) AS _&VNUM.,
              %END;
              %ELSE %DO;
                    STRIP("%SYSFUNC(VARNAME(&DSID,&VNUM.))") AS _&VNUM.,
              %END;
        %END;
        %* THIS LINE IS JUST FOR COVERING THE COMMA SIGN;
        MAX(%SYSFUNC(VARNAME(&DSID,&TVARNUM.))) AS _000
  FROM &DSIN.
  ;
  QUIT;

  DATA _NULL_;
        SET _OPTLEN;
        LENGTH _ATTRBUTE $32767;

        _ATTRBUTE='';
        %DO VNUM=1 %TO &TVARNUM.;
              _ATTRBUTE=CATX(',', _ATTRBUTE, _&VNUM.);
        %END;

        CALL SYMPUT("_VARATTIB", _ATTRBUTE);
  RUN;

  %*CREATE THE NEW DATASET;
  PROC SQL NOPRINT;
        CREATE TABLE &DSOUT.
              AS SELECT
                    &_VARATTIB.
              FROM &DSIN.
```

```
  QUIT;
%END;
```

Macro call example:

```
%let pooleddir1=&protpath/datasdtmplus/set1;
%LET STUDYPOOL1= lptsn1 lptsn3 lptsn2 lptsn6 lptsn10 lpts12 lpts13a
                 lpts13b lpts25 lpts28 lpts41 lpts55 lpts24 lpts87 lpts45
                 lpts52 lpts51 lpts100 lpts224 lpts164 lpts86;
%LET DOMAINS = CE CF CM DM DS EX PF PH PR RELREC SC SE SV;
%LET CHGVARLIST = CF(ATTCKNMB,CFAEMHSQ,CFNAMSEQ,EXIVDOSE,MONTHNMB,
                  STMLTNDY,WEEKNMB)|EX(EXDURDD);
%POOLDS(  domain_list      = &DOMAINS.
         ,library_list     = &STUDYPOOL1.
         ,pooled_outdir    = &pooleddir1.
         ,pooled_prefix    = i
         ,pooled_suffix    =
         ,pooled_info_excel= POOLED_INFO_A
         ,maxlen_char_var  = 2000
         ,Numeric_Var_Convert_List = %STR(&CHGVARLIST.)
         ,debug            = Y
         );
```

ADAM DATASET AUTOMATION DESIGN

During CSR generation process, creating ADaM dataset is very time consuming and often involves tedious work. Efficiently generating ADaM datasets has been a battle in statistical departments. In the past, there were discussions of automated ADaM dataset creation, and different approaches were investigated in the references list.

Building block programming is a prevalently used programming technique. In SAS, each macro function is actually a building block. With commonly used functional macro code, automation of ADaM datasets, especially safety related datasets can be automatic generated in most cases. Many programs are using this idea, but with plain SAS code instead of functional macro. The benefit of a centralized macro is obvious:

- Easy to maintain
- More robust
- Less error prone
- Flexible to insert or delete function
- Easy to trace the result

In order to illustrate the automation, let's look at some of the safety dataset generation flow charts (Figure 3):

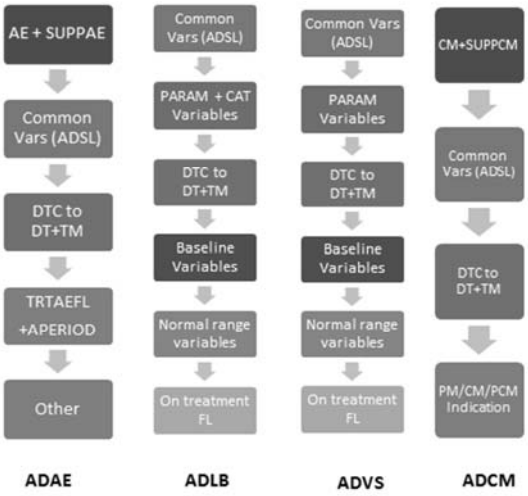

Figure 3 Flow chart of some safety related ADaM datasets programming.

ADAM AUTOMATION MODULES

From figure 3, there are many common functional blocks shared in multiple ADaM dataset generation flow charts. Based on building blocks design idea, the following small but focused macros were designed to facilitate ADaM dataset automation.

Macro SUPP_MRG: this macro is used for merging SDTM supplemental domain to main domain.

Syntax

```
%ISOtoDTM(ISODTC=, ADate=ADT, ATime=ATM, ADTF=ADTF, ATMF=ATMF, Rule=0);
```

Syntax

```
SUPP_MRG(DSIN=,  SUPP=, DSOUT=,  DOMAIN=, BYVAR=);
```

Arguments

DSIN: SDTM domain name. It is different in case the original dataset had been manipulated.

SUPP: Supplemental domain.

DSOUT: output dataset

DOMAIN: source domain name, if it is not DSIN.

BYVAR: Default is XXSEQ.

Calling example:

```
%SUPP_MRG(DSIN=AE, SUPP=, DSOUT=_AE_1, DOMAIN=);
```

Macro ISOtoDTM: This MACRO is target to convert ISO DTC format to numeric date and time. This macro is used in a data procedure.

Syntax

```
%ISOtoDTM(ISODTC=, ADate=ADT, ATime=ATM, ADTF=ADTF, ATMF=ATMF, Rule=0);
```

Arguments

ISODTC: input character data/time - IS8601DA or IS8601DT format.

ADate: output numeric data variable name

ATime: output numeric time variable name

ADTF: Date impute flag variable name

ATMF: Time impute flag variable name

Rule: imputation rule. 0: no imputation. 1: Impute to first possible date/time. 2: Impute to last possible date/time. 3. Impute to the date/time in the middle.

Calling example:

```
%ISOtoDTM(ISODTC=RFENDTC, ADate=RFENDT, ATime=RFENTM, Rule=0);
```

Macro BASEANVARS: This macro is using to generate basic ADaM variables: PARAM PARAMCD PARCAT1(NOT FOR VS) AVAL AVALC ADT ATM ADY.

Syntax

```
%BASEANVARS(DSIN=, DSOUT=, DOMAIN=);
```

Arguments

DSIN: input dataset.

DSOUT: output dataset

DOMAIN: source domain name

Calling example:

```
%BASEANVARS(DSIN=_LB_11, DSOUT=_LB_2, DOMAIN=LB);
```

Macro ANBLFL: This macro will generate the following list of variables: ANL01FL ANL01FN BASE ABLFL PSTBLFL AVISIT AVISITN CHG PCHG.

Syntax

```
%ANBLFL(DSIN=, DSOUT=, DOMAIN=, AVAL=, RULE=);
```

Arguments

DSIN: input dataset.

DSOUT: output dataset

DOMAIN: source domain name

AVAL: Analysis value

RULE: Base line select rule. 1: Select first in window. 2: default. select last in window. 3: using average in window.

Calling example:

```
%ANBLFL(DSIN=_LB_3, DSOUT=_LB_4, DOMAIN=LB);;
```

Macro HILO: This macro will generate the following list of variables: ANRLON ANRLO ANRHIN ANRHI ANRIND BNRIND;

Syntax

```
%HILO(DSIN=, DSOUT=, DOMAIN=);
```

Arguments

DSIN: input dataset.

DSOUT: output dataset

DOMAIN: source domain name

Macro ONTRTFL: Generate on treatment variables: ONTRTFL ONTRTFN ONTRT;

Syntax

```
%ONTRTFL(DSIN=, DSOUT=);
```

ADaM automation design:

With the necessary functional blocks ready, the automation of ADaM is ready to build. Here I did not include sponsor's own module, such as to fill in some missed lab value or it's special way to tackle a common issue. So the follow code is an example to build an ADaM dataset automation macro.

Here we show an example that using the designed macro, a draft ADLB automation program can be built as:

```
proc sort data=derived.adsl out=_adsl; by usubjid; run;
proc sort data=sdtm.lb out=lb; by usubjid lbseq; run;
data _lb_1;
  merge _adsl(in=a) lb(in=b);
  by usubjid;
  if a and b;
run;

*Generate variables: PARAM PARAMCD PARCAT1(LB) AVAL AVALC ADT ATM ADY;
%BASEANVARS(DSIN=_LB_1, DSOUT=_LB_2, DOMAIN=LB);

*Generate ADaM variables: ANL01FL ANL01FLN BASE ABLFL PSTBLFL CHG PCHG;
%ANBLFL_PRM(DSIN=_LB_2, DSOUT=_LB_3, DOMAIN=LB);


*HI/LO Variables: ANRLON ANRLO ANRHIN ANRHI ANRIND BNRIND;
%HILO(DSIN=_LB_3, DSOUT=_LB_4, DOMAIN=LB);

*ON TREATMENT FLAGS: ONTRTFL ONTRTFN ONTRT;
%ONTRTFL(DSIN=_LB_4, DSOUT=_LB_5);
```

CONCLUSION

Even though much effort has been put into ADaM dataset automation, true ADaM automation has not been widely applied. Along with the development of CDISC standards, the data integration and ADaM dataset automation is closer and closer.

REFERENCES

Reynolds, Natalia and Hibbetts, Keith. 2014. "Clinical Trial Data Integration: The Strategy, Benefits, and Logistics of Integrating Across a Compound". PharmaSUG 2014.

Steve Sibley. "Diving into Best Practices for Pooling Clinical Trial Data." 2016. https://www.certara.com.

Huang, Jianhua. 2013. "Macro %D_ADSL – Automating ADSL Creation from Metadata File" PharmaSUG 2013.

LaPann, Karin and Peterson, Terek. 2014. "Automation of ADAM Dataset Creation with a Retrospective, Prospective and Pragmatic Process". PharmaSUG 2014.

ACKNOWLEDGMENTS

The authors would like to thank Hal Li, Yong Zhu, Xiufeng Li, and Irene A Bobkoff from Merck & Co., Inc., Kenilworth, NJ, USA, for their statistical and programming support in this work.

RECOMMENDED READING

Base SAS® Procedures Guide

SAS® For Dummies®

CONTACT INFORMATION <HEADING 1>

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

William Wei
Merck & Co., Inc. USA
351 N Sumneytown Pike
North Wales, PA 19454
267-305-3277
<E-mail> william.wei@merck.com

Rinki Jajoo
Merck & Co., Inc. USA
126 E. Lincoln Ave
Rahway, NJ 07065
<E-mail> rinki_jajoo@merck.com

Susan Kramlik
Merck & Co., Inc. USA
351 N Sumneytown Pike
North Wales, PA 19454
<E-mail> susan.kramlik@merck.com