PharmaSUG 2019 - Paper BP-340

Programmatically mapping source variables to output SDTM (Study Data Tabulation Model) variables based upon entries in a standard specifications Excel® file workbook

Frederick Cieri CSG (Clinical Solutions Group);
Rama Arja MedImmune;
Ramesh Karuppusamy Covance

ABSTRACT

Based upon programmatically reading the entries from standard worksheets of a SDTM (Study Data Tabulation Model) specifications Excel file workbook, this paper details a macro application to map source variables to output SDTM variables. After learning how to properly fill out the Excel file, the macro should reduce errors and programming time by directly implementing the mappings of the source variables to the output variables. To create the output variables, the macro first reads the Excel file worksheets to create the metadata of the variable mappings. With the variable metadata, the macro reads the raw source data files, maps raw source variables to output variables by variable rename or format transformation, and creates an output data set of merged and appended raw source data files. For variable traceability, all the raw source variables will be in the output data set with the original name if the variable is not mapped, or with remapped name plus an added variable containing the original values if a format transformation occurs. For data set traceability, a variable is added to the output data set detailing the source data or data sets used to create the row. With the Excel file variable entries, the macro should be able to create about 50% or more of the variable outputs.

INTRODUCTION

The process of mapping raw source variables to SDTM variables could be done with a manual process, a hybrid process of part application and part manual, or a complete application approach. In the past as part of an all manual process, specifications were written into an Excel or Word file, and then the programmer manually copied the needed specifications into a SAS® program. The hybrid process programmatically reads an Excel file of specifications into a data set, and then the data set is used to dynamically write code in a SAS program to map the source variables. With the full application approach such as with the SAS Clinical Data Integration application, the application holds all the specifications, reads the specifications, and creates all the SDTM mapping and derivations. This paper will outline a hybrid approach to map the raw source variables to the SDTM variables.

DETAILS OF THE HYBRID PROCESS

With the hybrid process, the following steps are done to map the SDTM variables.

1. Read an Excel domain worksheet and create a data set of mappings
2. Read an Excel merge worksheet and create a data set of merge relationships.
3. Integrity checks.
4. Read each raw data set and remap.
5. Merge data sets.
6. Append and sort final output.
7. Display all actions performed by the macro in the log and listing.

STEP 1.) EXCEL DOMAIN WORKSHEET

For step 1 of reading in the domain specifications which will be used to derive the variable mappings, the SDTM specifications are placed into a standardized source mapping Excel workbook file, with a worksheet for each domain. Creating a standardized Excel file may be the most challenging part of the process because there are many scenarios that must be mapped properly by the user. Keep in mind that this process does not have the benefit of a graphical user interface to force the user to properly enter the values.

Using the AE (Adverse Events) domain, the columns of the standardized Excel domain worksheet are explained below and then examples are given in *Case 1.1) Character Output* and *Case 1.2) Numeric Output* below to explain the user entries below. The Excel sheet is then converted to a SAS data set with proc import.

Excel Columns:
Domain – domain name to identify SDTM domain

Keep_var – set to 'Y' to include the row in the SDTM data set. The value of 'N' causes the row to be ignored and not go to the SDTM data set.

Variable – output SDTM variable name

Source – list the raw source domains with variables separated by '.' delimiter to be mapped to the output variable with '|' delimiter separating each raw source domain with variable from each other. For example, 'AE.hlt' means hlt variable from AE raw source data. To transform the source variable, an optional format or informat can be associated with each raw source domain with variable.

Datatype – the variable output data type of 'Num' for numeric or 'Char' for character.

CASE 1.1) CHARACTER OUTPUT

Domain	KEEP_VAR	VARIABLE	SOURCE	DATATYPE
AE	Y	AEHLT	AE.hlt_char | AE2.hlt_char $50.| SFAE.hlt_num | SFAE2.hlt_num hltn. | SFAE3.hlt_num 5.0-L	Char

Table 1.1: Excel file mapping for character output.

With domain = 'AE', the SDTM domain is identified.

With keep_var = 'Y', the variable is created. If keep_var does not equal 'Y' then the row is ignored.

With variable = 'AEHLT', the output variable is named.

With datatype = 'Char', the output variable will be type character.

The source variable mapping syntax is explained below.
Source AE.hlt_char is character and gets mapped directly to AEHLT.
Source AE2.hlt_char is character and gets mapped to AEHLT using the $50 format.
Source SFAE.hlt_num is numeric and gets mapped to AEHLT by using the default best32 format with left alignment and missing values are converted to blanks.
Source SFAE2.hlt_num is numeric and gets mapped to AEHLT by using the hltn format catalogue format.
Source SFAE3.hlt_num is numeric and gets mapped to AEHLT by using the 5.0 format with left alignment.

No character length column is present in Case 1.1 because the SDTM character variable will be the maximum length of the source variables.

CASE 1.2) NUMERIC OUTPUT

Domain	KEEP_VAR	VARIABLE	SOURCE	DATATYPE
AE	Y	AEHLTCD	AE.hltc_num | SFAE.hltc_char | SFAESS.hltc_char hltc_c_i.	Num

Table 1.2: Excel file mapping for numeric output.

With variable = ‘AEHLTCD’, the output variable is named.

With datatype = ‘Num’, the output variable will be type numeric.

The source variable mapping syntax is explained below.

Source AE.hltc_num is numeric and gets mapped directly to AEHLTCD.
Source SFAE.hltc_char is character and gets mapped to AEHLTCD by using the default best32 informat.
Source SFAESS.hltc_char is character and gets mapped to AEHLTCD by using the hltc_c_i informat catalogue format.

STEP 2.) EXCEL MERGE WORKSHEET

For step two of reading in the merge relationships, this data entry is used to determine which raw source domains should be merged. To understand the user entries, the columns of the standardized Excel domain relationship worksheet are explained and then examples are given in *Case 2.1) Keep All the Rows* and *Case 2.2) Keep Selected Data* below with the AE domain. The Excel sheet is then converted to a SAS data set with proc import.

Domain – Name of domain to create by merging raw source domains. Links back to the SDTM domain worksheet.

KeyVariables– merge by variables separated by a space.

Source – list the raw source domains separated by ‘,’ to merge.

In Data set – Name of a raw source domain to determine the rows to keep. If blank all rows are output. Only one raw source domain value is allowed.

CASE 2.1) KEEP ALL THE ROWS

Domain	KeyVariables	SOURCE	In Data set
AE	Subject Aerefid	AE, AECD	

Table 2.1: Excel file of merge worksheet to keep all rows.

In Case 2.1 above, keep all the rows from raw source AE or AECD

CASE 2.2) KEEP SELECTED DATA

Domain	KeyVariables	SOURCE	In Data set
AE	Subject Aerefid	SFAE, SFAECD	SFAE

Table 2.2: Excel file of merge worksheet to keep selected rows.

In case 2.2 above, keep the rows from SFAE and rows in SFAECD that join to SFAE by the key variables

STEP 3.) INTEGRITY CHECKS

For step three of integrity checks, these checks involve SAS code consisting of about 50% of the total program to make sure the entries were properly made by the user in the Excel file and macro call. From my programming experience, integrity checks are essential in assisting the user to enter the proper information by giving the user messages in the log and listing file to correct entries. Some of the checks could have been made based upon data constraints within the Excel file, but for easier maintenance, all checks were done within the SAS program. Checks made by the CDISC (Clinical Data Interchange Standards Consortium) validator such as length of the SDTM variable name were not performed by the SAS program as this would be a duplication of effort. A sample of the checks involve items such as check macro parameters are filled out properly, Excel file is present, Excel worksheets are present, Excel columns are present, Excel columns are the proper type, Excel columns have the proper information, Excel rows do not have duplicates, raw source data and variables specified in the Excel domain and merge relationship worksheets are present, raw source data variables in the domain Excel worksheet are the correct type, formats in the domain Excel worksheet are present, merge by variables in the merge relationship Excel worksheet listed are compatible. An example of how to check an Excel file worksheet is present is listed in the code in *Case 3.1) Check Excel worksheet is present* below.

CASE 3.1) CHECK EXCEL WORKSHEET IS PRESENT

```
%macro mapping(file_parameter_name=,domain_parameter_name= );

 %local __z_exit_macro;
 %let   __z_exit_macro = 0;


 %*** Check file exists ***;
 data _null_;
       if fileexist(symget('file_parameter_name')) = 0 then
       do;
          put   'ER' 'ROR File not present ….';
          call symput('__z_exit_macro','1');
       end;
 run;

 %*** Abort no file. ***;
 %if &__z_exit_macro = 1 %then %goto exit;


 %*** Check if Excel worksheet with Libname to find worksheets ***;
  libname __z_inex xlsx "&file_parameter_name"   ;


 %*** Find domain worksheet. ***;
 proc sql feedback noprint;
  create table __z_excel_worksheet as
  select distinct
    memname
  from
    dictionary.tables
  where
   libname = '__Z_INEX' and
   memname = "%qupcase(&domain_parameter_name)"
```

```
     ;
    quit;


    %*** Check worksheet exists ***;
    data _null_;
       if 0 then set __z_excel_worksheet  nobs= __z_obs;

       if __z_obs = 0 then
          do;
             put   'ER' 'ROR Worksheet is not present ….';
             call symput('__z_exit_macro','1');
          end;

      stop;
    run;

    %*** Abort no worksheet. ***;
    %if &__z_exit_macro = 1 %then %goto exit;


    %*** SAS Code to derive mappings ***;


    %*** Abort  ***;
    %exit:

  %mend mapping;

  %mapping(file_parameter_name   = /path../SDTM_VARS.XLSX
           ,domain_parameter_name = AE   );
```

STEP 4.) READ AND MAP VARIABLES

For step four of reading in each raw data set and remap variables, every raw source domain listed in the domain worksheet explained in step one will have to be extracted and the variables remapped to the SDTM variables. With the format transformations mappings, each of the SDTM variables will have a variable added to trace the format transformation mappings. The key to the process is to transform the Excel domain worksheet into a usable SAS data set. Please follow the example with *Case 4.1) AE Mappings Excel* below for a subset of the mappings needed to create the AE domain below to understand the process.

CASE 4.1) AE MAPPINGS EXCEL

	A	B	C	D	E
1	**Domain**	**KEEP_VAR**	**VARIABLE**	**SOURCE**	**DATATYPE**
2	AE	Y	USUBJID	AE.subjid_raw_char | SFAE.subjid_raw_char | AECD.subjid_raw_char | SFAECD.subjid_raw_char | SFAESS.subjid_raw_char	Char

3	AE	Y	AEHLTCD	AE.hltc_num | SFAE.hltc_char | SFAESS.hltc_char hltc_c_i.	Num

Table 4.1: Excel file of AE domain mappings.

With SAS code, the Excel data in Case 4.1 becomes the SAS data set in *Case 4.2) SAS Data Set Mappings* below.

CASE 4.2) SAS DATA SET MAPPINGS

Obs	Domain	Variable	Source_ domain	Source_var	mapping	group
1	AE	USUBJID	AE	subjid_raw_char	USUBJID = subjid_raw_char;	1
2	AE	AEHLTCD	AE	hltc_num	AEHLTCD = hltc_num;	1
3	AE	USUBJID	AECD	subjid_raw_char	USUBJID = subjid_raw_char;	2
4	AE	USUBJID	SFAE	subjid_raw_char	USUBJID = subjid_raw_char;	3
5	AE	AEHLTCD	SFAE	hltc_char	AEHLTCD = input(hltc_char,best32.);	3
6	AE	USUBJID	SFAECD	subjid_raw_char	USUBJID = subjid_raw_char;	4
7	AE	USUBJID	SFAESS	subjid_raw_char	USUBJID = subjid_raw_char;	5
8	AE	AEHLTCD	SFAESS	hltc_char	AEHLTCD = input(hltc_char,hltc_c_i.);	5

Table 4.2: SAS data set of Excel file converted to data set mappings.

Through SAS code the data set in Case 4.2 is feed into the macro loop in *Case 4.3) SAS Macro Loop* and *Case 4.4) Sample SAS resolved macro code* below. Only part of the macro loop is displayed because the code would be too extensive. As a brief explanation, derive the number of distinct raw source domains into a macro variable which would be five for Case 4.2 as displayed in variable 'group'. Within the macro loop, subset SAS Data in case 4.2 to one raw source domain to extract. Derive macro variables for variable renames to avoid a SDTM variable overwriting an existing source variable, source data set name, copy variable code, format transformations, and drop variables copied or mapped to other variables. Place the derived macro variables into a data step for data extraction. In the example below, the macro code in Case 4.3 is followed by a sample data extraction in Case 4.4.

CASE 4.3) SAS MACRO LOOP

```
%*** Default format for numeric to character conversion using best32
format ***;
%*** with missing values converted to blanks. ***;
proc format;
 value __z_best32_miss
  .,.a-.z  = ' '
```

```
    other     = [best32.]
    ;
  run;


  %*** Derive number of extracts into macro variable &number_extracts.  ***;

  %*** Loop through each extract to bring in the data.  ***;
  %do &__z_i= 1 %to &number_extracts;

    %*** Derive source data ***;

    %*** Derive rename variables ***;

    %*** Derive variable to the identify the input source data ***;

    %*** Derive copy variables ***;

    %*** Derive variables format transformations ***;

    %*** Derive drop variables ***;

    %*** Extract ***;
    data __z_i_source_der_&__z_i;
      attrib &__z_i_source_dom_var_name length=$50 label='source domain';

      %*** Derive source data and rename variables ***;
      set &inlib..&__z_i_source_name_in.&__z_i_source_var_rename;

      %*** Identify the input source data name  ***;
      &__z_i_source_dom_var_name = "&__z_i_source_name_in";

      %*** copy variables ***;
      &__z_i_source_var_copy_syntax;

      %*** variables format transformations ***;
      &__z_i_derive_var_syntax;

      %*** drop variables ***;
      &__z_i_drop_var_syntax;
    run;

  %end;
```

CASE 4.4) SAMPLE SAS RESOLVED MACRO CODE

Macro code for loop run 1 (Case 4.2, group=1) to map SDTM variables from the AE raw source domain.

```
  data __z_i_source_der_1;

    attrib __Z_SOURCE_DOM_1_1 length=$50 label='source domain';

    %*** Derive source ***;
    %*** Rename source variable as SDTM variable would overwrite the value.
  ***;
    set rawin.AE( rename=(usubjid = usubjid_raw_char)); %*** Comment 4.4.1a
```

```
***;

  %*** Identify the input source domain ***;
  __Z_SOURCE_DOM_1_1 = "AE";

  %*** Copy variables for mapping and traceability for format
transformations ***;
  usubjid = subjid_raw_char; %*** Comment 4.4.1b ***;
  aehltcd = hltc_num;
  crit1_raw_num = crit1; %*** Comment 4.4.2a ***;
  crit2_raw_num = crit2;
  crit3_raw_num = crit3;
  crit4_raw_num = crit4;
  crit5_raw_num = crit5;
  crit6_raw_num = crit6;
  crit7_raw_num = crit7;
  maxupdated_raw_num = maxupdated;
  mincreated_raw_num = mincreated;
  recordid_raw_num = recordid;
  recordposition_raw_num = recordposition;
  subjectid_raw_num = subjectid;
  aeacn = action;
  aeacncrf = action;
  aeenrtpt = outcome;
  aeout = outcome;
  aeser = serious;

  %*** Variables format transformations ***;
  aesdth = put(crit1, __z_best32_miss.-L);  %*** Comment 4.4.2b ***;
  aerefid = put(recordid, __z_best32_miss.-L);
  aescong = put(crit7, __z_best32_miss.-L);
  aesdisab = put(crit5, __z_best32_miss.-L);
  aeshospp = put(crit4, __z_best32_miss.-L);
  aeshospr = put(crit3, __z_best32_miss.-L);
  aeslife = put(crit2, __z_best32_miss.-L);
  aesmie = put(crit6, __z_best32_miss.-L);
  aespid = put(recordposition, __z_best32_miss.-L);
  maxupdtc = put(maxupdated, __z_best32_miss.-L);
  mincrdtc = put(mincreated, __z_best32_miss.-L);
  subjectnew = put(subjectid, __z_best32_miss.-L);

  %*** Drop variables mapped or copied to other variables **; %* Comment
4.4.2c *;
  drop action aehltcd  crit1 crit2 crit3 crit4 crit5 crit6 crit7 hltc_num
       maxupdated mincreated outcome recordid recordposition serious
      subjectid subjid_raw_char;
run;
```

For the code in Case 4.4 above, to avoid a variable overwrite when a SDTM variable has the same name as a source variable, a variable rename is needed prior to any mapping code. For example, the raw source domain has the source variable USUBJID renamed in the set statement to trace variable USUBJID_RAW_CHAR in comment 4.4.1a because the output SDTM variable USUBJID is based upon the source variable SUBJID_RAW_CHAR in the copy code variable statements in comment 4.4.1b. For variable traceability with a format transformation, the source CRIT1 is copied to CRIT1_RAW_NUM in the copy code variables statements in comment 4.4.2a and then CRIT1 is transformed with a format numeric to character transformation with left alignment to create output variable AESDTH in the variable format transformation code statements in comment 4.4.2b. For drop of copied variables, the

CRIT1 variable is now dropped in comment 4.4.2c because the values of CRIT1 were preserved in CRIT1_RAW_NUM. In the *Case 4.5) SAS Data Set of Output Mapped Data* below, a partial display of the output mapping data is present showing how the SDTM output left aligned character variable AESDTH can be traced to the source CRIT1 numeric variable whose values are preserved in variable CRIT1_RAW_NUM.

CASE 4.5) SAS DATA SET OF OUTPUT MAPPED DATA

Obs	CRIT1_RAW_NUM	AESDTH
1	3	3
2	4	4
3	.	

Table 4.5: SAS data set of output mapped variable with trace to original values from renamed variable.

STEP 5.) MERGE DATA SETS

For step five of merge data sets, the raw source domains listed in the merge relationship worksheet which also must be present in the domain worksheet are merged as dictated by the entries. With the merge, each combined data set will have a variable to trace the raw source domains or domains that contributed to the merged row. The key to the process is to transform the Excel merge relationship worksheet into a usable SAS data set. Please follow the example with *Case 5.1) Excel Merge Relationship Worksheet* below of raw source AE actual term data to AE coded term data below to understand the process.

CASE 5.1) EXCEL MERGE RELATIONSHIP WORKSHEET

	A	B	C	D
1	**Domain**	**KeyVariables**	**Source**	**In Data set**
2	AE	Subject Aerefid	AE, AECD	
3	AE	Subject Aerefid	SFAE, SFAECD	SFAE

Table 5.1: Excel file of the merge relationships.

Through SAS code the Excel data in Case 5.1 becomes the SAS data set in *Case 5.2a) SAS Data Set of Merge Relationships* below.

CASE 5.2A) SAS DATA SET OF MERGE RELATIONSHIPS

Obs	Domain	KeyVariables	Source	In_data set	Merge_code	Reduce_code
1	AE	subject Aerefid	AE, AECD		merge ae aecd; by subject Aerefid;	
2	AE	subject Aerefid	SFAE,SFAECD	SFAE	merge sfae(in=in_sfae) sfaecd; by subject Aerefid;	if in_sfae;

Table 5.2a: SAS data set of merge relationships Excel file converted to merge meta-data code.

Through SAS code the Excel data in Case 5.1 is combined with step 4 of read and mapping which becomes the SAS data set in *Case 5.2b) SAS Data Set of Read and Remap* below. The 'map_source_data' variable gives the name of the new data set holding the extracted and mapped variables. The 'merge_group' variable gives the number of merge relationships. For variable 'merge_group' equal to 3 with raw source domain SFAESS in Case 5.2b, the raw source domain is not present in the Case 5.2a table of merge relationships, but is present in Case 4.1(AE mappings Excel). This means SFAESS is read in, renamed, and is not merged with another raw source domain. The 'merge_data_out' variable gives the resulting data set name from the merge.

CASE 5.2B) SAS DATA SET OF READ AND REMAP

Obs	Domain	Source	map_source_data	merge_group	merge_data_out
1	AE	AE	__Z_SOURCE_1_1	1	EXPAND_AE_1
2	AE	AECD	__Z_SOURCE_1_2	1	EXPAND_AE_1
3	AE	SFAE	__Z_SOURCE_2_1	2	EXPAND_AE_2
4	AE	SFAECD	__Z_SOURCE_2_2	2	EXPAND_AE_2
5	AE	SFAESS	__Z_SOURCE_3_1	3	EXPAND_AE_3

Table 5.2b: SAS data set of read and remap of source data names to temporary output map and merge SAS data set names.

Using Case 5.2b, the table in Case 5.2a is transformed to *Case 5.2c) SAS Data Set of Merge Syntax* below.

CASE 5.2C) SAS DATA SET OF MERGE SYNTAX

Obs	Domain	KeyVariables	merge_code	Reduce_code	merge_data_out	merge_group
1	AE	subject Aerefid	merge __Z_SOURCE_1_1 __Z_SOURCE_1_2; by subject Aerefid;		EXPAND_AE_1	1
2	AE	subject Aerefid	merge __Z_SOURCE_2_1(in=__z_2_1_in) __Z_SOURCE_2_2; by subject Aerefid;	if __z_2_1_in;	EXPAND_AE_2	2
3	AE		set __Z_SOURCE_3_1;		EXPAND_AE_3	3

Table 5.2c: SAS data set of merge syntax used to combine the source data.

The SAS code data sets in Cases 5.2b, and 5.2c are feed into the *Case 5.3) SAS Macro Loop* and *Case 5.4) Sample SAS Resolved Macro Code* below. In Case 5.2a, the 'merge_code' and 'reduce_code' variables use the original source data sets and variables to show traceability in the example, but the real input and output data set names are in Case 5.2b. Keep in mind every raw source domain was extracted and mapped in step four, and the merge of step five would use the extracted and mapped data which have names created by the macro. Only part of the macro loop is displayed because the code would be too extensive. As a brief explanation, derive the number of merge domain relationship into a macro variable which would be three for Case 5.2c present in variable 'merge_group'. Within the macro loop, subset SAS Data in case 5.2c with variable 'merge_group' to get the transformed raw source domains with merge syntax contained in variables 'merge_code' and 'reduce_code'. Derive macro variables for maximum length of each variable to avoid a merge warning of unequal lengths, merge syntax, reduce records, and identify raw source domain. Place the derived macro variables into a data step for the merge. In the example below, the code in Case 5.3 is followed by a sample extraction in Case 5.4.

CASE 5.3) SAS MACRO LOOP

```
%*** Derive number of merges into macro variable &number_merges.  ***;

%*** Loop through each merge.  ***;
%do &__z_i= 1 %to &number_merges;

  %*** Derive maximum length ***;

  %*** Derive merge syntax ***;

  %*** Derive reduce records ***;

  %*** Derive identify the raw source domain/s ***;

  data &__z_i_domain_name_group_name;
     attrib __source_domain length=$500 label='source domain';

     %*** Maximum length derived ***;
     length &__z_i_length_syntax;

     %*** Merge ***;
     &__z_i_merge_append_syntax;

     %*** Reduce records ***;
     &__z_i_in_keep_if_statement;

     %*** Identify the raw source domain/s ***;
     __source_domain = catx(', ',&__z_i_source_domains_var_names);

     %*** Drop temporary variables used to identify the raw source domains
***;
     drop %sysfunc(translate(%superq(__z_i_source_domains_var_names),
                                        %str( ),%str(,))) ;
  run;

%end;
```

CASE 5.4) SAMPLE SAS RESOLVED MACRO CODE

Macro code for loop run 2 (Case 5.2b and Case 5.2c, merge_group=2) of merge SFAE to SFAECD keeping records in SFAE or those in SFAECD that can be combined with SFAE using by variables SUBJECT and AEREFID.

```
data EXPAND_AE_2;
   attrib __source_domain length=$500 label='source domain';

   %*** Maximum length derived ***;
   length ACTION_STD 8 ADMDT 8 ADMDT_DD 8 ADMDT_INT 8 ADMDT_MM 8
          ADMDT_RAW $33 ADMDT_YYYY 8 AEACN $72 AEREFID $32
          …. Many more ….
```

```
            SITEID 8 SITENUMBER $150 STUDYID 8 STUDYSITEID 8
            SUBJECT $150 SUBJECTID_RAW_NUM 8
            __Z_SOURCE_DOM_2_1 $50 __Z_SOURCE_DOM_2_2 $50
         ;

    %*** Merge ***;
    merge __Z_SOURCE_2_1(in=__z_2_1_in) __Z_SOURCE_2_2;
    by SUBJECT AEREFID;

    %*** reduce records ***;
    If __z_2_1_in;

    %*** Identify the source domain/s.  ***;
    %** Variable __Z_SOURCE_DOM_2_1 derived in case 4.3 to equal source
 'SFAE'. **;
    %** Variable __Z_SOURCE_DOM_2_2 derived in case 4.3 to equal source
 'SFAECD'.**;
    __source_domain = catx(', ',__Z_SOURCE_DOM_2_1,__Z_SOURCE_DOM_2_2);

    %*** Drop temporary variables used to identify the source domains ***;
    drop __Z_SOURCE_DOM_2_1 __Z_SOURCE_DOM_2_2;
 run;
```

STEP 6.) APPEND

For step six of append and sort final output, the merged data and any non-merged extracted data get appended with the SAS set statement. The key to the process is to track all the data to append into a SAS data set. Please follow the examples below for append relationships to understand the process. For *Case 6.1) SAS Data Set of Append* below, the 'out_data' and 'sort_by' values were macro parameters input by the user.

CASE 6.1) SAS DATA SET OF APPEND

Obs	Domain	merge_data_out	Out_data	sort_by
1	AE	EXPAND_AE_1	AE_OUT	Subject aerefid
2	AE	EXPAND_AE_2	AE_OUT	Subject aerefid
3	AE	EXPAND_AE_3	AE_OUT	Subject aerefid

Table 6.1: SAS data set of data to append to create the final output.

Through SAS code the data set in Case 6.1 is feed into the macro for *Case 6.2) SAS Macro Code Append* and *Case 6.3) Sample SAS Resolved Macro Code* below. As a brief explanation, derive macro variables for data sets to be appended, and maximum length of each variable to avoid a warning of unequal lengths. Place the derived macro variables into a data step for the append join. In the example below, the code is followed by a sample extraction.

CASE 6.2) SAS MACRO CODE APPEND

```
 %**********************************************************************
   Append all extract groups
   &__z_final_app_length_syntax - length syntax to avoid truncation warning
   &__z_final_set_append - data set names for final append
 **********************************************************************;
 data __z_append_extract_groups;
```

```
   length &__z_final_app_length_syntax;
   set &__z_final_set_append ;
 run;


 %*** Output final sorted data. &sortby- sort by variables ***;
 proc sort data=__z_append_extract_groups out=&outdata;
     by &sortby ;
 run;
```

CASE 6.3) SAMPLE SAS RESOLVED MACRO CODE

```
 %*** Append all extract groups ***;
 data __z_append_extract_groups;

    %*** Maximum length derived ***;
    length ACTION_STD 8 ADMDT 8 ADMDT_DD 8 ADMDT_INT 8 ADMDT_MM 8
           ADMDT_RAW $33 ADMDT_YYYY 8 AEACN $72  AEREFID $32
           …. Many more ….
           SITEID 8 SITENUMBER $150 STUDYID 8 STUDYSITEID 8
           SUBJECT $150 SUBJECTID_RAW_NUM 8
           __Z_SOURCE_DOM_1_1 $50 __Z_SOURCE_DOM_1_2 $50
        ;
    %*** append ***;
    set EXPAND_AE_1 EXPAND_AE_2 EXPAND_AE_3;

 run;


 %*** Output final sorted data ***;
 proc sort data=__z_append_extract_groups out=AE_OUT;
   by  subject aerefid ;
 run;
```

CASE 6.4) FREQUENCY OF THE TRACE OF MERGE RELATIONSHIPS

```
                             The FREQ Procedure

 __SOURCE_                                     Cumulative    Cumulative
 DOMAIN          Frequency      Percent         Frequency       Percent
-----------------------------------------------------------------------
 AE                     15         0.10              15          0.10
 AE, AECD            14362        96.38           14377         96.48
 AECD                    5         0.03           14382         96.51
 SFAE                   38         0.25           14420         96.77
 SFAE, SFAECD          400         2.68           14822         99.45
 SFAESS                 80         0.54           14902        100.00
```

Listing 6.4: Frequency of the trace of merge relationships

After the append in Case 6.1, 6.2, and 6.3, Listing 6.4 above shows how the trace variable __SOURCE _DOMAIN lists the name of the raw source domain for the origin of the row. To explain, the yellow rows in Listing 6.4 link to Case 5.1 merge relationships Excel row 2 where the row can come from raw source AE or AECD. The green rows in Listing 6.4 link to Case 5.1 merge relationships Excel row 3 where the row must come from raw source SFAE or where raw source SFAECD can be combined to SFAE. The blue row shows raw source SFAESS is not merged with any other data and traces back to the raw source domains listed in the variable mappings in Case 4.1 AE mappings.

STEP 7.) DISPLAY OF LOG AND LISTING

For step seven of display all actions performed by the macro in the log and listing, every mapped variable, merged data set and append are detailed in the log and listing for the user to trace the macro. The key to the process is to use all the metadata in steps 1-6 containing data sets for the Excel worksheets, mappings, merges, and append. Please view *Case 7.1) Display Mapping Details in Log* below for an example of a log display of mappings. The text comments highlighted in yellow are to explain the details of the display and are not part of the log.

CASE 7.1) DISPLAY MAPPING DETAILS IN LOG

```
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - Variable derivation details.  (7.1a)
M_SDTM_MAP Macro - File parameter spec_path: ..path/SDTM IG.XLSX
M_SDTM_MAP Macro - Worksheet parameter domain: ae
M_SDTM_MAP Macro - Worksheet parameter domain merge worksheet: Merge Raw Data
M_SDTM_MAP Macro - Library parameter inlib: rawin
M_SDTM_MAP Macro - Final output data set parameter outdata: ae_out
M_SDTM_MAP Macro - Sort by Variables: subject aerefid

    1.)  Excel row=7, VARIABLE=AEREFID, DATATYPE=CHAR   (7.1b)
       source column=Ae.Recordid | SFAe.Recordid | AeCD.Recordid | SFAeCD.Recordid

       Source domain=AE, Source variable=RECORDID, Source variable type=NUM  (7.1c)

       Derivation details->
Source variable copied due to variable type conflict with output variable.
Set output variable equal to the data type converted source variable.
Drop source variable.

        Action-> Copy         Syntax-> recordid_raw_num=recordid;  (7.1d)
        Action-> Derive       Syntax-> aerefid=put(recordid,__z_best32_miss.-L);  (7.1e)
        Action-> Drop         Syntax-> recordid  (7.1f)
```

To explain the mappings in Case 7.1, a macro banner in comment 7.1a shows the parameters entered by the user. This mapping occurred because the output SDTM variable AEREFID in comment 7.1b is character, but the source AE.RECORDID (AE is the raw source domain, RECORDID is the variable) in comment 7.1c is numeric. Accordingly, the original source AE.RECORDID in comment 7.1d is copied into trace output variable RECORDID_RAW_NUM. Afterwards, output AEREFID in comment 7.1e is derived with the transformed numeric to character left aligned value with put function on source numeric variable AE.RECORDID with macro created format __z_best32_miss which converts numeric missing to character blanks and the rest of the values use best32 format. Finally, the source RECORDID variable in comment 7.1f is removed from the output data as the value is preserved in output variable RECORDID_RAW_NUM in comment 7.1d. The output RECORDID_RAW_NUM variable can now be used to verify the AEREFID derivation.

After the mappings, the log displays the merges and appends as is displayed in *Case 7.2) Display Merge and Append Details in Log* below. The text highlighted in yellow is to explain the details of the display and is not part of the log.

CASE 7.2) DISPLAY MERGE AND APPEND DETAILS IN LOG

```
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - Process flow to create output data set.  (7.2a)
M_SDTM_MAP Macro - File parameter spec_path: ..Path/SDTM IG.XLSX
M_SDTM_MAP Macro - Worksheet parameter domain: ae
M_SDTM_MAP Macro - Worksheet parameter domain merge worksheet: Merge Raw Data
M_SDTM_MAP Macro - Library parameter inlib: rawin
M_SDTM_MAP Macro - Final output data set parameter outdata: aeraw_out
M_SDTM_MAP Macro - Sort by Variables: subject  aerefid
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - 1.) Data manipulation: Merge source domains group (7.2b)
M_SDTM_MAP Macro -      Worksheet: Merge Raw Data      Excel row: 2
M_SDTM_MAP Macro -      Output: EXPAND_AE_1
M_SDTM_MAP Macro -      Source domains: AE, AECD
M_SDTM_MAP Macro -      Key Variables: SUBJECT AEREFID
M_SDTM_MAP Macro -      Keep Records: All
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - 2.) Data manipulation: Merge source domains group (7.2c)
M_SDTM_MAP Macro -      Worksheet: Merge Raw Data      Excel row: 3
M_SDTM_MAP Macro -      Output: EXPAND_AE_2
M_SDTM_MAP Macro -      Source domains: SFAE, SFAECD
M_SDTM_MAP Macro -      Key Variables: SUBJECT AEREFID
M_SDTM_MAP Macro -      Keep Records: SFAE
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - 3.) Data manipulation: Read (7.2d)
M_SDTM_MAP Macro -      Worksheet: AE
M_SDTM_MAP Macro -      Output: EXPAND_AE_3
M_SDTM_MAP Macro -      Source domain: SFAESS
M_SDTM_MAP Macro -
M_SDTM_MAP Macro - All the data above are appended into data set ae_out. (7.2e)
M_SDTM_MAP Macro -
```

To explain the merge and append details in Case 7.2, a macro banner in comment 7.2a shows the parameters entered by the user. Starting the merge relationships in comment 7.2b, the raw source AE actual term data and AECD coded term data are merged by the key variables keeping all the records from either of the data sets. In comment 7.2c, the raw source SFAE actual term data and SFAECD coded term data are merged by the key variables keeping all records from SFAE and records from SFAECD that can be combined to SFAE. In comment 7.2d, the raw source SFAESS data is not merged but is displayed to show the data is to be appended. In comment 7.2e, all the data sets in 7.2b, 7.2c, and 7.2d are then appended.

CONCLUSION

To summarize, this paper showed how to map raw source variables to SDTM variables by reading in Excel worksheets of SDTM mappings and merge relationships, converting the Excel worksheets to data sets, and deriving SAS macro code to read the data sets, create the SDTM mappings, merge the data, and append the data. The benefit of the macro is the direct mapping of Excel requirements which should lead to fewer discrepancies and faster development. The mapping and format transformations outlined in the paper can create about 50-70% of the variables for many of the SDTM domains. For the rest of the output variables, post processing of the SDTM mapping data set would be needed to derive all the derivations. To increase the number of mappings, a large format catalogue of raw data to SDTM mappings is essential such as to create ‘yes/no’ output from ‘1’, or ‘0’ values, number months from text months, AE severity decode from AE code, lab test decode from lab test code, and others. For the future to get closer to creating 100% of the SDTM output, the process could be enhanced to add where clauses, data set options, complex joins of data, and mapping and transformation macros. For faster validation of the mappings, the metadata created by the mapping macro could be used to automate frequency checks of format transformations as all the relevant information of source variable, format, and new variable are contained in the metadata. To conclude, the process of getting the final SDTM mappings can almost be completely automated if the development team adds Excel worksheets and SAS code to achieve the goal.

CONTACT INFORMATION

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

Frederick Cieri

Systems and macro development programmer

CSG (Clinical Solutions Group)

cierif@MedImmune.com

Rama Arja

Senior Principal Statistical Programmer

MedImmune

arjar@MedImmune.com

Ramesh Karuppusamy

Statistical Programmer

Covance

karuppusamyr@MedImmune.com