CDISC SDTM Mapping Engine

Bas van Bakel, OCS Life Sciences, ‘s-Hertogenbosch, the Netherlands

ABSTRACT

This paper describes a mapping framework that has been implemented at a large pharmaceutical company in which internally maintained tools are generated that describe and execute source-to-target mappings in a structured way.
The concept of the mapping framework is easy to understand and, because of its modular structure, the framework can be implemented with minimum effort.

INTRODUCTION

During the process of generating SDTM data from CDASH-based operational data or other source datasets many data mappings take place. Companies usually have tools available that help in the process of defining and applying these data mappings in order to ensure the SDTM data is generated and it is clearly described how this is done.
Tools can be provided by external parties and even though that may have its advantages it might be advantageous to have a company tool available that is not maintained by an external party, but internally.

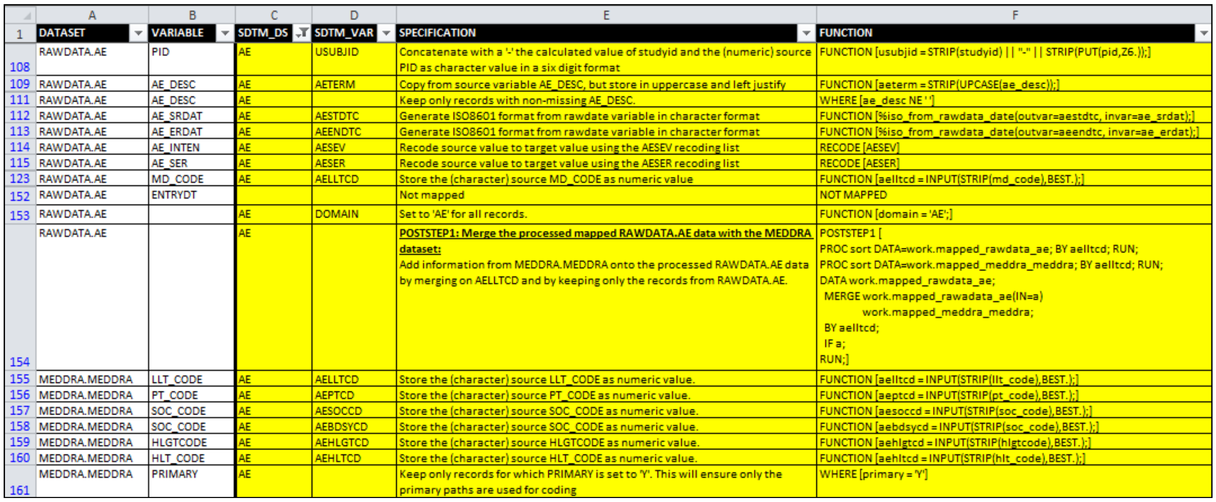
This paper describes a concept that has been implemented at a large pharmaceutical company in which such an internally maintained tool is generated. Within this concept Microsoft Excel is used to describe, in a very structured way, both the human readable source-to-target specifications, as well as the machine-executable SAS code.
The paper will not only show the general idea behind the process, but will also have more technical sections that focus on part of the SAS code that is used within this process.

THE SCOPE

The examples that are used in this paper focus on generating SDTM from source data, but the framework can also be applied on other sources and other targets as long as the targets are defined in a structured way and their structure can be obtained by the framework.
Within this paper it will be explained how the framework makes use of the ‘target metadata’ to obtain the expected (trial specific) structure of the target (e.g. the attributes (type, length, etcetera) of the variables of the SDTM domains) and how it ensures the output is aligned with it. Defining and maintaining the metadata of the targets will not be in scope of this paper.

THE BASICS

One of the most important aspects of the framework is that there is a **single** Microsoft Excel spreadsheet that contains **all** source-to-target specifications and, directly next to them, the translation of these specifications into code or pseudo-code. An example of this mapping specification document is shown below (some columns and rows are hidden). The white columns contain the source datasets and variables. The yellow columns contain the target datasets and variables and the specifications and (pseudo-)code to convert the sources to the targets.



	A	B	C	D	E	F
1	DATASET	VARIABLE	SDTM_DS	SDTM_VAR	SPECIFICATION	FUNCTION
108	RAWDATA.AE	PID	AE	USUBJID	Concatenate with a '-' the calculated value of studyid and the (numeric) source PID as character value in a six digit format	FUNCTION [usubjid = STRIP(studyid) || "-" || STRIP(PUT(pid,Z6.));]
109	RAWDATA.AE	AE_DESC	AE	AETERM	Copy from source variable AE_DESC, but store in uppercase and left justify	FUNCTION [aeterm = STRIP(UPCASE(ae_desc));]
111	RAWDATA.AE	AE_DESC	AE		Keep only records with non-missing AE_DESC.	WHERE [ae_desc NE ' ']
112	RAWDATA.AE	AE_SRDAT	AE	AESTDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=aestdtc, invar=ae_srdat);]
113	RAWDATA.AE	AE_ERDAT	AE	AEENDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=aeendtc, invar=ae_erdat);]
114	RAWDATA.AE	AE_INTEN	AE	AESEV	Recode source value to target value using the AESEV recoding list	RECODE [AESEV]
115	RAWDATA.AE	AE_SER	AE	AESER	Recode source value to target value using the AESER recoding list	RECODE [AESER]
123	RAWDATA.AE	MD_CODE	AE	AELLTCD	Store the (character) source MD_CODE as numeric value	FUNCTION [aelltcd = INPUT(STRIP(md_code),BEST.);]
152	RAWDATA.AE	ENTRYDT			Not mapped	NOT MAPPED
153	RAWDATA.AE		AE	DOMAIN	Set to 'AE' for all records.	FUNCTION [domain = 'AE';]

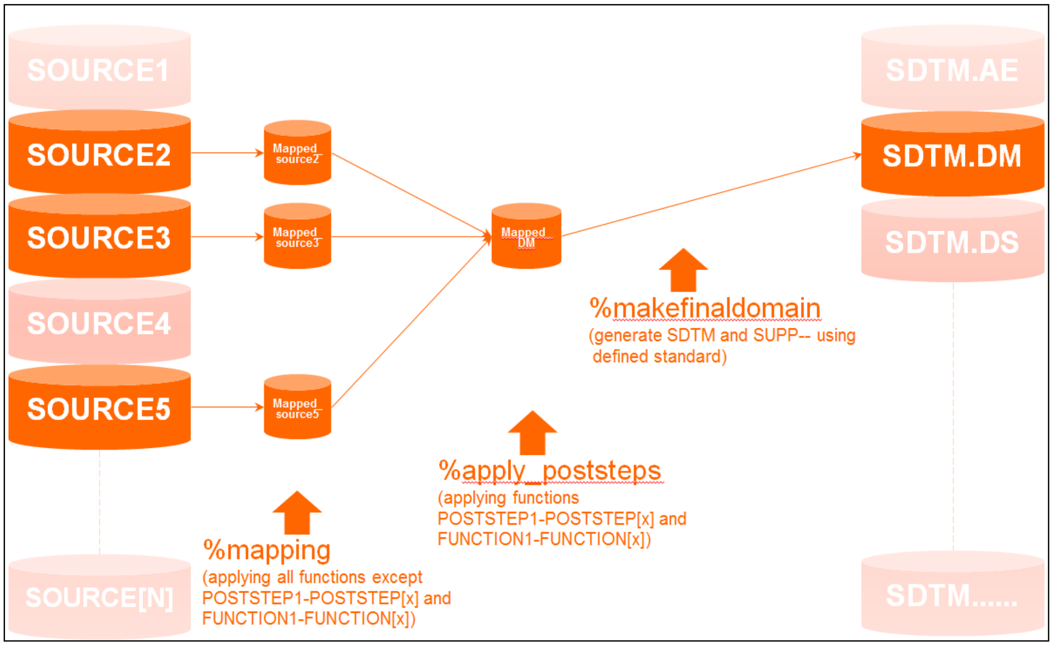
154	RAWDATA.AE		AE		**POSTSTEP1: Merge the processed mapped RAWDATA.AE data with the MEDDRA dataset:**
					Add information from MEDDRA.MEDDRA onto the processed RAWDATA.AE data by merging on AELLTCD and by keeping only the records from RAWDATA.AE.
						POSTSTEP1 [
						PROC sort DATA=work.mapped_rawdata_ae; BY aelltcd; RUN;
						PROC sort DATA=work.mapped_meddra_meddra; BY aelltcd; RUN;
						DATA work.mapped_rawdata_ae;
						MERGE work.mapped_rawadata_ae(IN=a)
						work.mapped_meddra_meddra;
						BY aelltcd;
						IF a;
						RUN;]

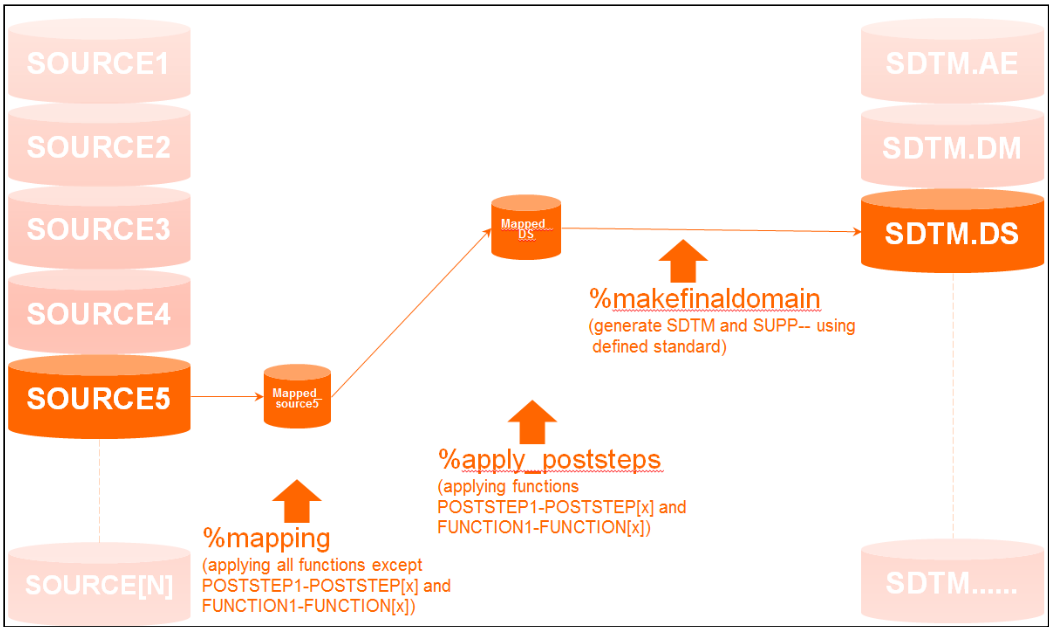
155	MEDDRA.MEDDRA	LLT_CODE	AE	AELLTCD	Store the (character) source LLT_CODE as numeric value.	FUNCTION [aelltcd = INPUT(STRIP(llt_code),BEST.);]
156	MEDDRA.MEDDRA	PT_CODE	AE	AEPTCD	Store the (character) source PT_CODE as numeric value.	FUNCTION [aeptcd = INPUT(STRIP(pt_code),BEST.);]
157	MEDDRA.MEDDRA	SOC_CODE	AE	AESOCCD	Store the (character) source SOC_CODE as numeric value.	FUNCTION [aesoccd = INPUT(STRIP(soc_code),BEST.);]
158	MEDDRA.MEDDRA	SOC_CODE	AE	AEBDSYCD	Store the (character) source SOC_CODE as numeric value.	FUNCTION [aebdsycd = INPUT(STRIP(soc_code),BEST.);]
159	MEDDRA.MEDDRA	HLGTCODE	AE	AEHLGTCD	Store the (character) source HLGTCODE as numeric value.	FUNCTION [aehlgtcd = INPUT(STRIP(hlgtcode),BEST.);]
160	MEDDRA.MEDDRA	HLT_CODE	AE	AEHLTCD	Store the (character) source HLT_CODE as numeric value.	FUNCTION [aehltcd = INPUT(STRIP(hlt_code),BEST.);]
161	MEDDRA.MEDDRA	PRIMARY	AE		Keep only records for which PRIMARY is set to 'Y'. This will ensure only the primary paths are used for coding	WHERE [primary = 'Y']

Macros are available that translate the (pseudo-)code into SAS code, execute that SAS code in a specific order and output the target dataset with the attributes as available in the ‘target metadata’. Independent of the SDTM domain that is to be generated there will always be three macros that are executed in sequence to facilitate this.

- %mapping: This macro will first determine (per SDTM output domain) which source datasets are needed. Then each of the source datasets are converted into intermediate ‘mapped’ dataset(s) by using some of the (pseudo-) code as specified in the mapping specification document.
- %apply_poststeps: After the ‘mapped’ dataset(s) are generated by the %mapping macro the remaining (pseudo)-code is used to further process and combine the ‘mapped’ dataset(s) into one single intermediate dataset.
- %makefinaldomain: After all generated SAS code is executed and all data is combined into one dataset, the SDTM dataset and Supplemental Qualifier dataset are generated by aligning the datasets and variables with the attributes as specified in the ‘target metadata’.

The pictures below are examples of the above mentioned process when generating the SDTM Demographics domain from three source datasets and generating the SDTM Disposition domain from a single source dataset.
Please note that, independent of the source dataset(s) that are used, the concept remains the same.


SOURCE1
SOURCE2
SOURCE3
SOURCE4
SOURCE5
SOURCE[N]
Mapped source2
Mapped source3
Mapped source5
Mapped DM
SDTM.AE
SDTM.DM
SDTM.DS
SDTM......
%makefinaldomain
(generate SDTM and SUPP-- using defined standard)
%apply_poststeps
(applying functions POSTSTEP1-POSTSTEP[x] and FUNCTION1-FUNCTION[x])
%mapping
(applying all functions except POSTSTEP1-POSTSTEP[x] and FUNCTION1-FUNCTION[x])


SOURCE1
SOURCE2
SOURCE3
SOURCE4
SOURCE5
SOURCE[N]
Mapped source5
Mapped DS
SDTM.AE
SDTM.DM
SDTM.DS
SDTM......
%makefinaldomain
(generate SDTM and SUPP-- using defined standard)
%apply_poststeps
(applying functions POSTSTEP1-POSTSTEP[x] and FUNCTION1-FUNCTION[x])
%mapping
(applying all functions except POSTSTEP1-POSTSTEP[x] and FUNCTION1-FUNCTION[x])

THE DETAILS

This section will more thoroughly describe the functions of each of the three macros and how they translate the information from the mapping specification document into executable SAS code and subsequently execute this SAS code to generate the SDTM domains.

THE CODE-GENERATING MACROS

The %mapping macro and the %apply_poststeps macro read the (pseudo-)code specified in the FUNCTION column of the mapping specification document, translate that to SAS code and execute that SAS code in a specific order to generate the intermediate (near-final) dataset that is fed to the %makefinaldomain macro.

Depending on the function that is specified in the FUNCTION column the SAS code is executed either when the source data is converted to the 'mapped' datasets (%mapping) or when post processing and combining of the 'mapped' datasets take place (%apply_poststeps). The following functions are available and are described in detail in the following paragraphs:

- %mapping: 'WHERE', 'RENAME', 'COPY', 'FUNCTION', 'RECODE', 'STACK1-STACK[n]', 'KEEP'.
- %apply_poststeps: 'POSTSTEP1-POSTSTEP[n]', 'FUNCTION1-FUNCTION[n]'.

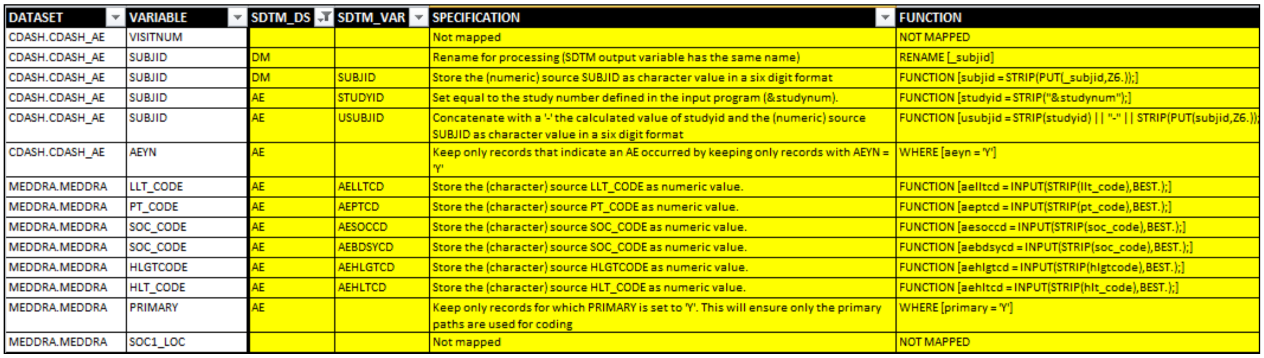
THE MAPPING MACRO

Each source dataset for which any row is available that is marked as being needed to generate the SDTM domain will be read by the %mapping macro. Each of these source datasets will generate one output 'mapped' dataset. Only the source variables that are marked as being needed in this process are read from the source datasets.

For example, in the mapping specification document below, when the AE domain is to be generated, the CDASH.CDASH_AE and MEDDRA.MEDDRA source datasets are read because they are both marked as sources for the SDTM AE domain (as indicated by 'AE' in the column SDTM_DS).

When reading the CDASH.CDASH_AE dataset only the variables AEYN and SUBJID will be kept.

When reading the MEDDRA.MEDDRA dataset only the variables LLT_CODE, PT_CODE, SOC_CODE, HLGTCODE, HLT_CODE and PRIMARY will be kept.



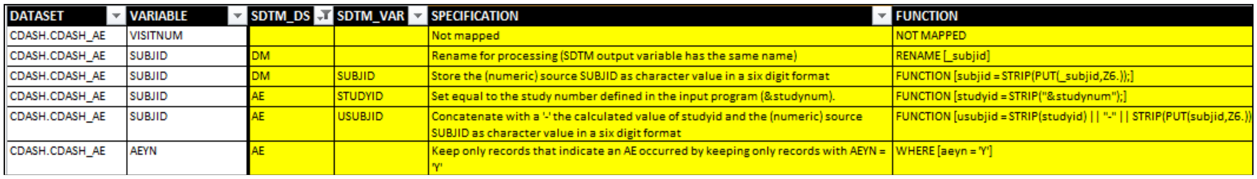
DATASET	VARIABLE	SDTM_DS	SDTM_VAR	SPECIFICATION	FUNCTION
CDASH.CDASH_AE	VISITNUM			Not mapped	NOT MAPPED
CDASH.CDASH_AE	SUBJID	DM		Rename for processing (SDTM output variable has the same name)	RENAME [_subjid]
CDASH.CDASH_AE	SUBJID	DM	SUBJID	Store the (numeric) source SUBJID as character value in a six digit format	FUNCTION [subjid = STRIP(PUT(_subjid,Z6.));]
CDASH.CDASH_AE	SUBJID	AE	STUDYID	Set equal to the study number defined in the input program (&studynum).	FUNCTION [studyid = STRIP("&studynum");]
CDASH.CDASH_AE	SUBJID	AE	USUBJID	Concatenate with a '-' the calculated value of studyid and the (numeric) source SUBJID as character value in a six digit format	FUNCTION [usubjid = STRIP(studyid) || "-" || STRIP(PUT(subjid,Z6.));
CDASH.CDASH_AE	AEYN	AE		Keep only records that indicate an AE occurred by keeping only records with AEYN = 'Y'	WHERE [aeyn = 'Y']
MEDDRA.MEDDRA	LLT_CODE	AE	AELLTCD	Store the (character) source LLT_CODE as numeric value.	FUNCTION [aelltcd = INPUT(STRIP(llt_code),BEST.);]
MEDDRA.MEDDRA	PT_CODE	AE	AEPTCD	Store the (character) source PT_CODE as numeric value.	FUNCTION [aeptcd = INPUT(STRIP(pt_code),BEST.);]
MEDDRA.MEDDRA	SOC_CODE	AE	AESOCCD	Store the (character) source SOC_CODE as numeric value.	FUNCTION [aesoccd = INPUT(STRIP(soc_code),BEST.);]
MEDDRA.MEDDRA	SOC_CODE	AE	AEBDSYCD	Store the (character) source SOC_CODE as numeric value.	FUNCTION [aebdsycd = INPUT(STRIP(soc_code),BEST.);]
MEDDRA.MEDDRA	HLGTCODE	AE	AEHLGTCD	Store the (character) source HLGTCODE as numeric value.	FUNCTION [aehlgtcd = INPUT(STRIP(hlgtcode),BEST.);]
MEDDRA.MEDDRA	HLT_CODE	AE	AEHLTCD	Store the (character) source HLT_CODE as numeric value.	FUNCTION [aehltcd = INPUT(STRIP(hlt_code),BEST.);]
MEDDRA.MEDDRA	PRIMARY	AE		Keep only records for which PRIMARY is set to 'Y'. This will ensure only the primary paths are used for coding	WHERE [primary = 'Y']
MEDDRA.MEDDRA	SOC1_LOC			Not mapped	NOT MAPPED

Variables that are not needed in the conversion process are marked with specification and function "NOT MAPPED". This process ensures that you cannot use variables that you have not marked in the specifications as being needed in the conversion process.

Once the source datasets are read, the functions in the FUNCTION column are translated to SAS code and applied when reading the source datasets.

The function 'WHERE'

When the function "WHERE [statement]" is specified the exact where clause as specified within the brackets will be applied on the source dataset. If there are multiple WHERE-clauses specified they will both be applied on the source dataset so only records fulfilling both clauses will be kept when reading the source dataset.



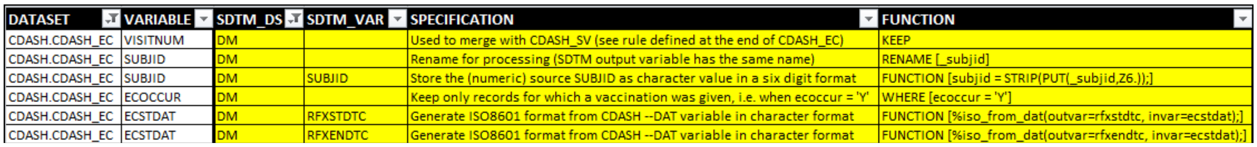
DATASET	VARIABLE	SDTM_DS	SDTM_VAR	SPECIFICATION	FUNCTION
CDASH.CDASH_AE	VISITNUM			Not mapped	NOT MAPPED
CDASH.CDASH_AE	SUBJID	DM		Rename for processing (SDTM output variable has the same name)	RENAME [_subjid]
CDASH.CDASH_AE	SUBJID	DM	SUBJID	Store the (numeric) source SUBJID as character value in a six digit format	FUNCTION [subjid = STRIP(PUT(_subjid,Z6.));]
CDASH.CDASH_AE	SUBJID	AE	STUDYID	Set equal to the study number defined in the input program (&studynum).	FUNCTION [studyid = STRIP("&studynum");]
CDASH.CDASH_AE	SUBJID	AE	USUBJID	Concatenate with a '-' the calculated value of studyid and the (numeric) source SUBJID as character value in a six digit format	FUNCTION [usubjid = STRIP(studyid) || "-" || STRIP(PUT(subjid,Z6.))
CDASH.CDASH_AE	AEYN	AE		Keep only records that indicate an AE occurred by keeping only records with AEYN = 'Y'	WHERE [aeyn = 'Y']

The example above will result in the following where clause to be applied on the source dataset:

```
WHERE=((aeyn = 'Y'))
```

The function 'RENAME'

When the function "RENAME [statement]" is specified the exact renaming clause as specified in the brackets will be applied on the source dataset using the source variable for which it is specified. (e.g. RENAME=([source variable] = [value in brackets after rename]).



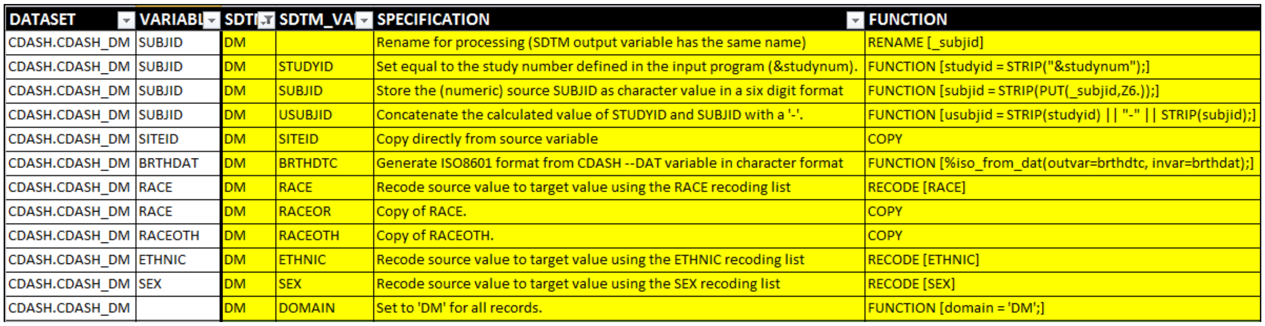
DATASET	VARIABLE	SDTM_DS	SDTM_VAR	SPECIFICATION	FUNCTION
CDASH.CDASH_EC	VISITNUM	DM		Used to merge with CDASH_SV (see rule defined at the end of CDASH_EC)	KEEP
CDASH.CDASH_EC	SUBJID	DM		Rename for processing (SDTM output variable has the same name)	RENAME [_subjid]
CDASH.CDASH_EC	SUBJID	DM	SUBJID	Store the (numeric) source SUBJID as character value in a six digit format	FUNCTION [subjid = STRIP(PUT(_subjid,Z6.));]
CDASH.CDASH_EC	ECOCCUR	DM		Keep only records for which a vaccination was given, i.e. when ecoccur = 'Y'	WHERE [ecoccur = 'Y']
CDASH.CDASH_EC	ECSTDAT	DM	RFXSTDTC	Generate ISO8601 format from CDASH --DAT variable in character format	FUNCTION [%iso_from_dat(outvar=rfxstdtc, invar=ecstdat);]
CDASH.CDASH_EC	ECSTDAT	DM	RFXENDTC	Generate ISO8601 format from CDASH --DAT variable in character format	FUNCTION [%iso_from_dat(outvar=rfxendtc, invar=ecstdat);]

The example above will result in the following rename statement to be applied on the source dataset:

```
RENAME=(subjid=_subjid)
```

The function 'COPY'

When the function "COPY" is specified the source variable will be copied to the SDTM target variable by generating a SAS statement in the format [target variable] = [source variable];. If there are multiple COPY-functions specified for a source dataset they will be concatenated: [target variable1] = [source variable1]; [target variable2] = [source variable2]; [target variable3] = [source variable3];



DATASET	VARIABL	SDTI	SDTM_VA	SPECIFICATION	FUNCTION
CDASH.CDASH_DM	SUBJID	DM		Rename for processing (SDTM output variable has the same name)	RENAME [_subjid]
CDASH.CDASH_DM	SUBJID	DM	STUDYID	Set equal to the study number defined in the input program (&studynum).	FUNCTION [studyid = STRIP("&studynum");]
CDASH.CDASH_DM	SUBJID	DM	SUBJID	Store the (numeric) source SUBJID as character value in a six digit format	FUNCTION [subjid = STRIP(PUT(_subjid,Z6.));]
CDASH.CDASH_DM	SUBJID	DM	USUBJID	Concatenate the calculated value of STUDYID and SUBJID with a '-'.	FUNCTION [usubjid = STRIP(studyid) || "-" || STRIP(subjid);]
CDASH.CDASH_DM	SITEID	DM	SITEID	Copy directly from source variable	COPY
CDASH.CDASH_DM	BRTHDAT	DM	BRTHDTC	Generate ISO8601 format from CDASH --DAT variable in character format	FUNCTION [%iso_from_dat(outvar=brthdtc, invar=brthdat);]
CDASH.CDASH_DM	RACE	DM	RACE	Recode source value to target value using the RACE recoding list	RECODE [RACE]
CDASH.CDASH_DM	RACE	DM	RACEOR	Copy of RACE.	COPY
CDASH.CDASH_DM	RACEOTH	DM	RACEOTH	Copy of RACEOTH.	COPY
CDASH.CDASH_DM	ETHNIC	DM	ETHNIC	Recode source value to target value using the ETHNIC recoding list	RECODE [ETHNIC]
CDASH.CDASH_DM	SEX	DM	SEX	Recode source value to target value using the SEX recoding list	RECODE [SEX]
CDASH.CDASH_DM		DM	DOMAIN	Set to 'DM' for all records.	FUNCTION [domain = 'DM';]

The example above will result in the following code which is executed inside a data step reading the source dataset and outputting to the 'mapped' dataset:

```
SITEID = SITEID; RACEOR = RACE; RACEOTH=RACEOTH;
```

The function 'FUNCTION'

When the function "FUNCTION" is specified the exact code as specified in brackets will be executed. If there are multiple FUNCTION functions specified for a source dataset they will be concatenated in the order of which they are available in the mapping specification document. This means it is possible to refer to variables derived in other functions as long as those other functions are mentioned at a higher location in the mapping specification document.

The previous example contains five FUNCTION statements. They will lead to the following code which is executed inside a data step reading the source dataset and outputting to the ‘mapped’ dataset:

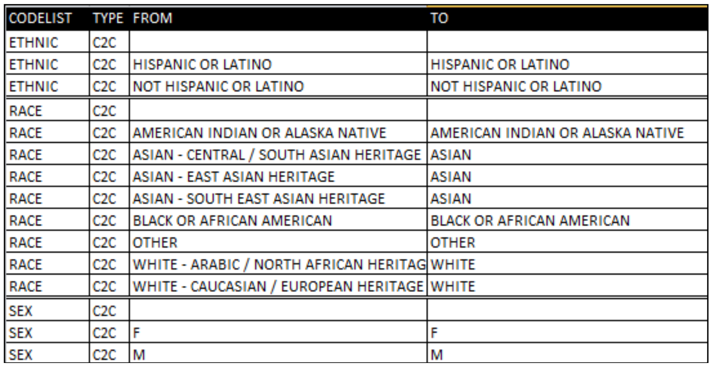
```
studyid = STRIP("&studynum");
subjid = STRIP(PUT(_subjid,Z6.);
usubjid = STRIP(studyid) || "-" || STRIP(subjid);
%iso_from_dat(outvar=brthdtc, invar=brthdat);
domain = ‘DM’;
```

The function ‘RECODE’

When the function “RECODE” is specified the value specified in brackets will be used to check a specific part of the mapping specification document for a recoding list. IF-THEN-ELSE code will be generated to translate source values to SDTM values making use of that list. If there are multiple RECODE functions specified for a source dataset they will be concatenated.

The previous example contains three RECODE statements referring to recoding lists ‘ETHNIC’, ‘RACE’ and ‘SEX’.

Suppose that the following recoding list is available in the mapping specification:



CODELIST	TYPE	FROM	TO
ETHNIC	C2C		
ETHNIC	C2C	HISPANIC OR LATINO	HISPANIC OR LATINO
ETHNIC	C2C	NOT HISPANIC OR LATINO	NOT HISPANIC OR LATINO
RACE	C2C		
RACE	C2C	AMERICAN INDIAN OR ALASKA NATIVE	AMERICAN INDIAN OR ALASKA NATIVE
RACE	C2C	ASIAN - CENTRAL / SOUTH ASIAN HERITAGE	ASIAN
RACE	C2C	ASIAN - EAST ASIAN HERITAGE	ASIAN
RACE	C2C	ASIAN - SOUTH EAST ASIAN HERITAGE	ASIAN
RACE	C2C	BLACK OR AFRICAN AMERICAN	BLACK OR AFRICAN AMERICAN
RACE	C2C	OTHER	OTHER
RACE	C2C	WHITE - ARABIC / NORTH AFRICAN HERITAG	WHITE
RACE	C2C	WHITE - CAUCASIAN / EUROPEAN HERITAGE	WHITE
SEX	C2C		
SEX	C2C	F	F
SEX	C2C	M	M

Then this will lead to the following code which is executed inside a data step reading the source dataset and outputting to the ‘mapped’ dataset: (please note that ‘C2C’ is an abbreviation for ‘Character-to-Character’ and ensures that the type of the source and target variables are correctly set):

```
IF RACE = '' THEN RACE = '';
ELSE IF RACE = 'AMERICAN INDIAN OR ALASKA NATIVE' THEN RACE = 'AMERICAN INDIAN OR ALASKA NATIVE';
ELSE IF RACE = 'ASIAN – CENTRAL / SOUTH ASIAN HERITAGE' THEN RACE = 'ASIAN';
ELSE IF RACE = 'ASIAN – EAST ASIAN HERITAGE' THEN RACE = 'ASIAN';
ELSE IF RACE = 'ASIAN – SOUTH EAST ASIAN HERITAGE' THEN RACE = 'ASIAN';
ELSE IF RACE = 'BLACK OR AFRICAN AMERICAN' THEN RACE = 'BLACK OR AFRICAN AMERICAN';
ELSE IF RACE = 'OTHER' THEN RACE = 'OTHER';
ELSE IF RACE = 'WHITE – ARABIC / NORTH AFRICAN HERITAGE' THEN RACE = 'WHITE';
ELSE IF RACE = 'WHITE = CAUCASIAN / EUROPEAN HERITAGE' THEN RACE = 'WHITE';

IF ETHNIC = '' THEN ETHNIC = '';
ELSE IF ETHNIC = 'HISPANIC OR LATINO' THEN ETHNIC = 'HISPANIC OR LATINO';
ELSE IF ETHNIC = 'NOT HISPANIC OR LATINO' THEN ETHNIC = 'NOT HISPANIC OR LATINO';

IF SEX = '' THEN SEX = '';
ELSE IF SEX = 'F' THEN SEX = 'F';
ELSE IF SEX = 'M' THEN SEX = 'M';
```

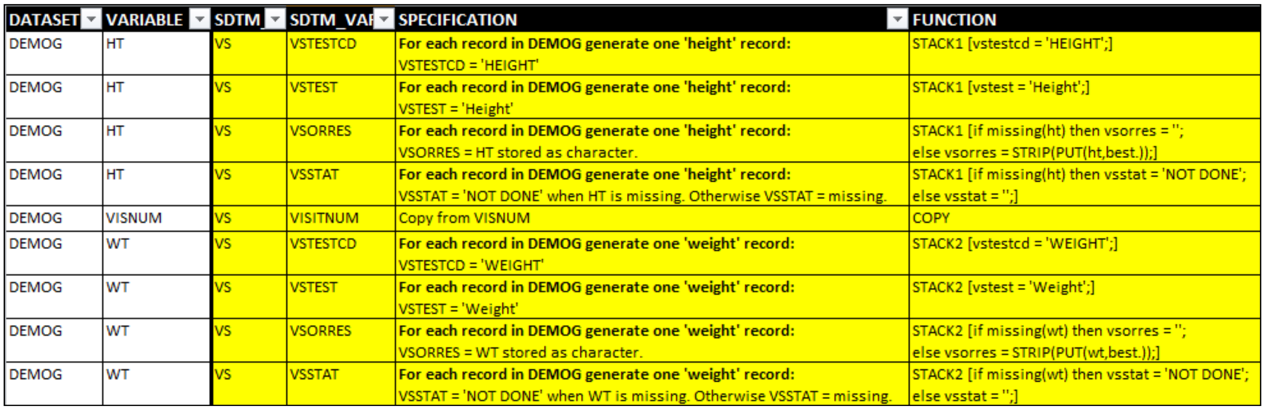
Next to the above statements the following statements are generated that give warnings if a source variable contains a value that is not available in the recoding list:

```
IF RACE NOT IN ('', 'AMERICAN INDIAN OR ALASKA NATIVE', 'ASIAN – CENTRAL / SOUTH ASIAN HERITAGE', 'ASIAN – EAST ASIAN
HERITAGE', 'ASIAN – SOUTH EAST ASIAN HERITAGE', 'BLACK OR AFRICAN AMERICAN', 'BLACK OR AFRICAN AMERICAN', 'OTHER', 'WHITE –
ARABIC / NORTH AFRICAN HERITAGE', 'WHITE = CAUCASIAN / EUROPEAN HERITAGE') THEN PUT "WAR" "NING: Variable RACE has value "
[…] "not in recoding list";

IF ETHNIC NOT IN ('', 'HISPANIC OR LATINO', 'NOT HISPANIC OR LATINO') THEN PUT "WAR" "NING: Variable ETHNIC has value " […]
"not in recoding list";

IF SEX NOT IN ('', 'F', 'M') THEN PUT "WAR" "NING: Variable SEX has value " […] "not in recoding list";
```

The functions ‘STACK[1]-STACK[n]’

When the function “STACK1”, “STACK2” or “STACK[n]” is specified the exact function as specified in brackets will be executed. If there are multiple STACK[x] functions specified for a source dataset they will be concatenated in the order of which they are available in the mapping specification document. This means it is possible to refer to variables derived in other functions as long as those other functions are mentioned at a higher location in the mapping specification document.

In principle STACK works in the same way as FUNCTION except that after each set of statements with the same stack number (STACK[x]) an OUTPUT statement will be executed. In practice the stack functions will therefore be used to **generate multiple output records from a single source record**.



DATASET	VARIABLE	SDTM_	SDTM_VAF	SPECIFICATION	FUNCTION
DEMOG	HT	VS	VSTESTCD	**For each record in DEMOG generate one 'height' record:** VSTESTCD = 'HEIGHT'	STACK1 [vstestcd = 'HEIGHT';]
DEMOG	HT	VS	VSTEST	**For each record in DEMOG generate one 'height' record:** VSTEST = 'Height'	STACK1 [vstest = 'Height';]

DEMOG	HT	VS	VSORRES	**For each record in DEMOG generate one 'height' record:**
				VSORRES = HT stored as character.
					STACK1 [if missing(ht) then vsorres = '';
					else vsorres = STRIP(PUT(ht,best.));]

DEMOG	HT	VS	VSSTAT	**For each record in DEMOG generate one 'height' record:**
				VSSTAT = 'NOT DONE' when HT is missing. Otherwise VSSTAT = missing.
					STACK1 [if missing(ht) then vsstat = 'NOT DONE';
					else vsstat = '';]

DEMOG	VISNUM	VS	VISITNUM	Copy from VISNUM	COPY
DEMOG	WT	VS	VSTESTCD	**For each record in DEMOG generate one 'weight' record:** VSTESTCD = 'WEIGHT'	STACK2 [vstestcd = 'WEIGHT';]
DEMOG	WT	VS	VSTEST	**For each record in DEMOG generate one 'weight' record:** VSTEST = 'Weight'	STACK2 [vstest = 'Weight';]

DEMOG	WT	VS	VSORRES	**For each record in DEMOG generate one 'weight' record:**
				VSORRES = WT stored as character.
					STACK2 [if missing(wt) then vsorres = '';
					else vsorres = STRIP(PUT(wt,best.));]

DEMOG	WT	VS	VSSTAT	**For each record in DEMOG generate one 'weight' record:**
				VSSTAT = 'NOT DONE' when WT is missing. Otherwise VSSTAT = missing.
					STACK2 [if missing(wt) then vsstat = 'NOT DONE';
					else vsstat = '';]

The example above will result in the following code which is executed inside a data step reading the source dataset and outputting to the ‘mapped’ dataset:

```
vstestcd = 'HEIGHT';
vstest = 'Height';
if missing(ht) then vsorres = '';
else vsorres = STRIP(PUT(ht,best.));
if missing(ht) then vsstat = 'NOT DONE';
else vsstat = '';
OUTPUT; /*End of STACK1*/

vstestcd = 'WEIGHT';
vstest = 'Weight';
if missing(wt) then vsorres = '';
else vsorres = STRIP(PUT(wt,best.));
if missing(wt) then vsstat = 'NOT DONE';
else vsstat = '';
OUTPUT; /*End of STACK2*/
```

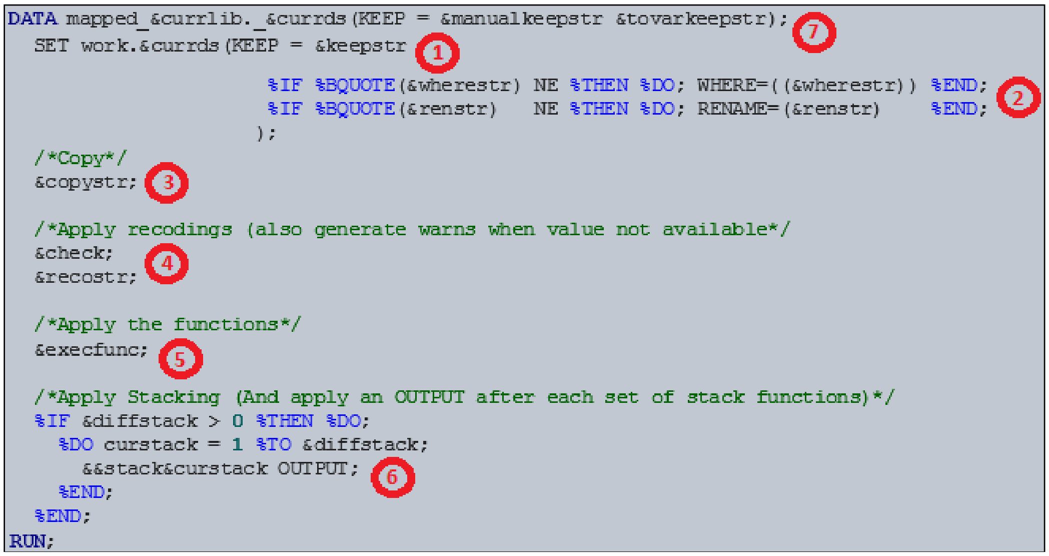
The function ‘KEEP’

The %mapping macro will, by default, only keep the variables from the mapped datasets that are marked as target SDTM variables. However, it might be that a source variable is needed to generate SDTM output, but that it is not used by the %mapping macro itself, but by the %apply_poststeps macro that is called afterwards.

If you specify the function ‘KEEP’ the variable from the source dataset will be retained in the ‘mapped’ version of the dataset and can be referenced in the code that is used by the %apply_poststeps macro.

The overview of the functions

To get a better understanding of the order in which the code is executed please find a (very simplified) version of the data step that is generated by the %mapping macro below.



```
DATA mapped_&currlib._&currds(KEEP = &manualkeepstr &tovarkeepstr);
  SET work.&currds(KEEP = &keepstr

                                  %IF %BQUOTE(&wherestr) NE %THEN %DO; WHERE=((&wherestr)) %END;
                                  %IF %BQUOTE(&renstr)   NE %THEN %DO; RENAME=(&renstr)    %END;
                                 );
  /*Copy*/
  &copystr;

  /*Apply recodings (also generate warns when value not available*/
  &check;
  &recostr;

  /*Apply the functions*/
  &execfunc;

  /*Apply Stacking (And apply an OUTPUT after each set of stack functions)*/
  %IF &diffstack > 0 %THEN %DO;
    %DO curstack = 1 %TO &diffstack;
      &&stack&curstack OUTPUT;
    %END;
  %END;
RUN;
```

(1) When reading the source dataset only the variables that are needed (as specified in the mapping specification document) are kept

(2) The WHERE-clause(s) is/are applied to select only the appropriate records and the RENAME function(s) is/are applied to rename variables.

(3) The COPY functions are executed in the order as available in the mapping specification document.

(4) The RECODE functions (and the code to check for unexpected values) are executed in the order as available in the mapping specification document.

(5) The FUNCTION functions are executed in the order as available in the mapping specification document.

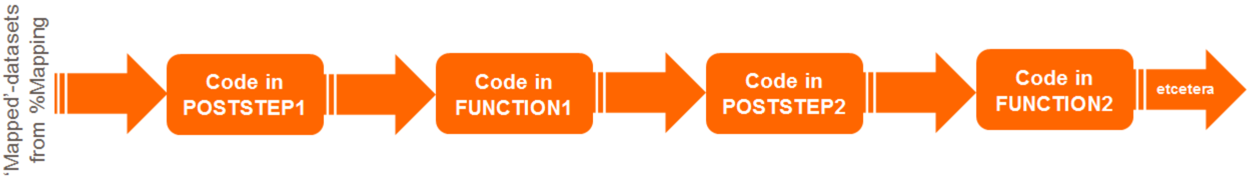
(6) If available, the STACK1 functions are executed in the order as available in the mapping specification document and afterwards an OUTPUT statement is executed. If available the STACK2 functions are executed in the order as available in the mapping specification document and afterwards an OUTPUT statement is executed. (same logic applies for STACK3 up to STACK[n])

(7) The 'mapped' dataset is generated, but only variables that are marked as output variables or marked as being needed for post-processing (i.e. with the KEEP function) are kept.

THE APPLY_POSTSTEPS MACRO

The %apply_poststeps macro is used to combine the 'mapped' datasets and apply post-processing on the combined datasets. The result of the macro is a single (near-final) dataset that is fed to the %makefinaldomain macro which generates the final SDTM domain.

The %apply_poststeps macro is used to obtain the code specified within the POSTSTEP1-POSTSTEP[x] and FUNCTION1-FUNCTION[x] functions of the mapping specification document and is executed after the intermediate 'mapped' datasets are generated by the %mapping macro. It will execute the POSTSTEP[x] and FUNCTION[x] functions in the following order:



The functions ‘POSTSTEP[1]- POSTSTEP[n]’

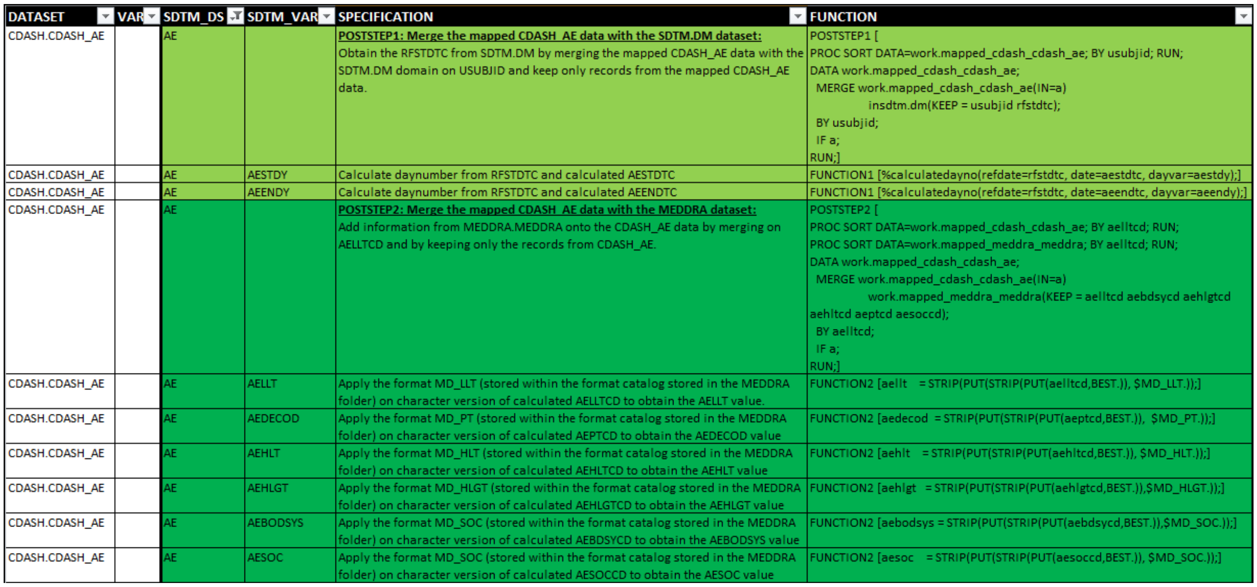
When a function like “POSTSTEP[x]” is specified the exact SAS code as specified in brackets will be executed. Contrary to the SAS code that is generated by the %mapping macro, the code within the POSTSTEP will not be executed within a DATA STEP’. Therefore this makes POSTSTEP functions ideal for merging multiple source datasets or for SQL statements.

The functions ‘FUNCTION[1]- FUNCTION[n]’

Functions like “FUNCTION[x]” will be handled in exactly the same way as the FUNCTION as used in the %mapping macro and will be executed within a DATA STEP. The only difference is the timing of when the code is executed:

- “FUNCTION” will be read by the %mapping macro and the code is executed when reading the source dataset and generating the ‘mapped‘ dataset.
- “FUNCTION[x]” will be read by the %apply_poststeps macro and the code is executed on the output of “POSTSTEP[x]”.

For example, in the mapping specification document below two “POSTSTEP[x]” and two “FUNCTION[x]” are available:



DATASET	VAR	SDTM_DS	SDTM_VAR	SPECIFICATION	FUNCTION

CDASH.CDASH_AE		AE		**POSTSTEP1: Merge the mapped CDASH_AE data with the SDTM.DM dataset:**
				Obtain the RFSTDTC from SDTM.DM by merging the mapped CDASH_AE data with the SDTM.DM domain on USUBJID and keep only records from the mapped CDASH_AE data.
					POSTSTEP1 [
					PROC SORT DATA=work.mapped_cdash_cdash_ae; BY usubjid; RUN;
					DATA work.mapped_cdash_cdash_ae;
					MERGE work.mapped_cdash_cdash_ae(IN=a)
					insdtm.dm(KEEP = usubjid rfstdtc);
					BY usubjid;
					IF a;
					RUN;]

CDASH.CDASH_AE		AE	AESTDY	Calculate daynumber from RFSTDTC and calculated AESTDTC	FUNCTION1 [%calculatedayno(refdate=rfstdtc, date=aestdtc, dayvar=aestdy);]
CDASH.CDASH_AE		AE	AEENDY	Calculate daynumber from RFSTDTC and calculated AEENDTC	FUNCTION1 [%calculatedayno(refdate=rfstdtc, date=aeendtc, dayvar=aeendy);]

CDASH.CDASH_AE		AE		**POSTSTEP2: Merge the mapped CDASH_AE data with the MEDDRA dataset:**
				Add information from MEDDRA.MEDDRA onto the CDASH_AE data by merging on AELLTCD and by keeping only the records from CDASH_AE.
					POSTSTEP2 [
					PROC SORT DATA=work.mapped_cdash_cdash_ae; BY aelltcd; RUN;
					PROC SORT DATA=work.mapped_meddra_meddra; BY aelltcd; RUN;
					DATA work.mapped_cdash_cdash_ae;
					MERGE work.mapped_cdash_cdash_ae(IN=a)
					work.mapped_meddra_meddra(KEEP = aelltcd aebdsycd aehlgtcd aehltcd aeptcd aesoccd);
					BY aelltcd;
					IF a;
					RUN;]

CDASH.CDASH_AE		AE	AELLT	Apply the format MD_LLT (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AELLTCD to obtain the AELLT value.	FUNCTION2 [aellt = STRIP(PUT(STRIP(PUT(aelltcd,BEST.)), $MD_LLT.));]
CDASH.CDASH_AE		AE	AEDECOD	Apply the format MD_PT (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AEPTCD to obtain the AEDECOD value	FUNCTION2 [aedecod = STRIP(PUT(STRIP(PUT(aeptcd,BEST.)), $MD_PT.));]
CDASH.CDASH_AE		AE	AEHLT	Apply the format MD_HLT (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AEHLTCD to obtain the AEHLT value	FUNCTION2 [aehlt = STRIP(PUT(STRIP(PUT(aehltcd,BEST.)), $MD_HLT.));]
CDASH.CDASH_AE		AE	AEHLGT	Apply the format MD_HLGT (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AEHLGTCD to obtain the AEHLGT value	FUNCTION2 [aehlgt = STRIP(PUT(STRIP(PUT(aehlgtcd,BEST.)),$MD_HLGT.));]
CDASH.CDASH_AE		AE	AEBODSYS	Apply the format MD_SOC (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AEBDSYCD to obtain the AEBODSYS value	FUNCTION2 [aebodsys = STRIP(PUT(STRIP(PUT(aebdsycd,BEST.)),$MD_SOC.));]
CDASH.CDASH_AE		AE	AESOC	Apply the format MD_SOC (stored within the format catalog stored in the MEDDRA folder) on character version of calculated AESOCCD to obtain the AESOC value	FUNCTION2 [aesoc = STRIP(PUT(STRIP(PUT(aesoccd,BEST.)), $MD_SOC.));]

(Please note that the records that are read by the %mapping macro and executed to generate the mapped_cdash_cdash_ae dataset are not shown)

This will lead to the following code:

```
/*POSTSTEP1: Code executed as in specification document*/
PROC SORT DATA=work.mapped_cdash_cdash_ae; BY usubjid; RUN;
DATA work.mapped_cdash_cdash_ae;
  MERGE work.mapped_cdash_cdash_ae(IN=a)
        insdtm.dm(KEEP = usubjid rfstdtc);
  BY usubjid;
  IF a;
RUN;

/*FUNCTION1: Code executed inside a data step*/
DATA work.mapped_cdash_cdash_ae;
  SET work.mapped_cdash_cdash_ae;
  %calculatedayno(refdate=rfstdtc, date=aestdtc, dayvar=aestdy);
  %calculatedayno(refdate=rfstdtc, date=aeendtc, dayvar=aeendy);
RUN;

/*POSTSTEP2: Code executed as in specification document */
PROC SORT DATA=work.mapped_cdash_cdash_ae; BY aelltcd; RUN;
PROC SORT DATA=work.mapped_meddra_meddra; BY aelltcd; RUN;
DATA work.mapped_cdash_cdash_ae;
  MERGE work.mapped_cdash_cdash_ae(IN=a)
        work.mapped_meddra_meddra(KEEP = aelltcd aebdsycd aehlgtcd aehltcd aeptcd aesoccd);
  BY aelltcd;
  IF a;
RUN;
```

```
/*FUNCTION2: Code executed inside a data step*/
DATA work.mapped_cdash_cdash_ae;
  SET work.mapped_cdash_cdash_ae;
  aellt    = STRIP(PUT(STRIP(PUT(aelltcd,BEST.)), $MD_LLT.));
  aedecod  = STRIP(PUT(STRIP(PUT(aeptcd,BEST.)),  $MD_PT.));
  aehlt    = STRIP(PUT(STRIP(PUT(aehltcd,BEST.)), $MD_HLT.));
  aehlgt   = STRIP(PUT(STRIP(PUT(aehlgtcd,BEST.)),$MD_HLGT.));
  aebodsys = STRIP(PUT(STRIP(PUT(aebdsycd,BEST.)),$MD_SOC.));
  aesoc    = STRIP(PUT(STRIP(PUT(aesoccd,BEST.)), $MD_SOC.));
RUN;
```

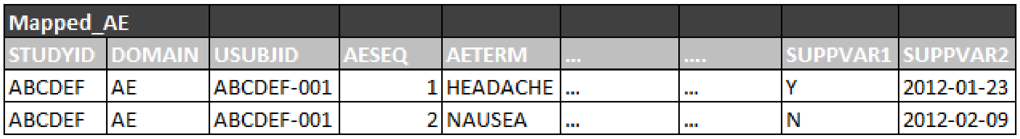
It is possible that there is only one source dataset needed for a SDTM domain and that no post processing is needed. In that case there will not be any “POSTSTEP[x]” functions defined and the output ‘mapped’ dataset of the %mapping macro will be directly fed to the %makefinaldomain macro.

THE MAKEFINALDOMAIN MACRO

The final macro that is executed before the SDTM dataset is generated is the macro %makefinaldomain.
Using the near-final SDTM dataset that is generated by the %mapping and %apply_poststeps macros and the ‘target metadata’ the macro %makefinaldomain will ensure that:

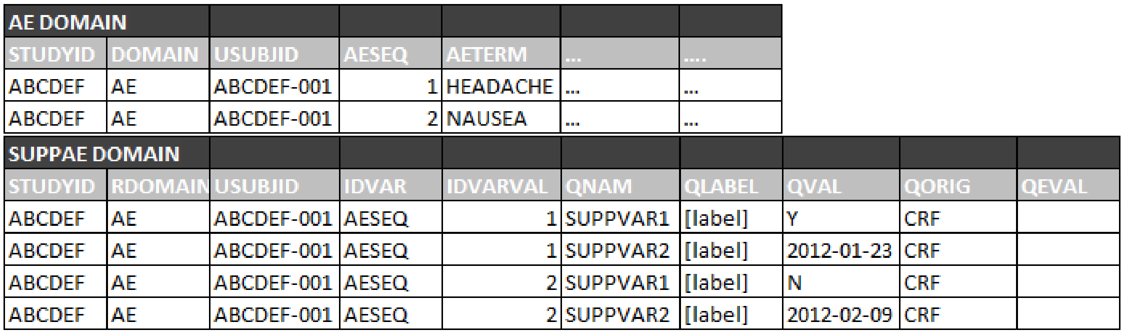
- Only variables that are defined in the ‘target metadata’ are kept in the output;
- Variables and domains get the attributes (type, length, label) as defined in the ‘target metadata’;
- Domains are sorted on the keys as defined in the ‘target metadata’ and the --SEQ variable is generated using those keys;
- The intermediate dataset is split into a SDTM ‘main’ domain and a Supplemental Qualifier domain (e.g. split into AE and SUPPAE).

When entering the source-to-target specifications and the (pseudo-)code in the mapping specification spreadsheet it is important to take the above functionality into account.
For example, you can add variables that actually need to go into the Supplemental Qualifier domain to the intermediate dataset ‘main’ domain itself (see SUPPVAR1 and SUPPVAR2):



Mapped_AE								
STUDYID	DOMAIN	USUBJID	AESEQ	AETERM	...	....	SUPPVAR1	SUPPVAR2
ABCDEF	AE	ABCDEF-001	1	HEADACHE	...	...	Y	2012-01-23
ABCDEF	AE	ABCDEF-001	2	NAUSEA	...	...	N	2012-02-09

The %makefinaldomain macro will ensure that the AE and SUPPAE domains are generated and that these variables are moved to the Supplemental Qualifier domain:



AE DOMAIN						
STUDYID	DOMAIN	USUBJID	AESEQ	AETERM	...	....
ABCDEF	AE	ABCDEF-001	1	HEADACHE	...	...
ABCDEF	AE	ABCDEF-001	2	NAUSEA	...	...

SUPPAE DOMAIN									
STUDYID	RDOMAIN	USUBJID	IDVAR	IDVARVAL	QNAM	QLABEL	QVAL	QORIG	QEVAL
ABCDEF	AE	ABCDEF-001	AESEQ	1	SUPPVAR1	[label]	Y	CRF	
ABCDEF	AE	ABCDEF-001	AESEQ	1	SUPPVAR2	[label]	2012-01-23	CRF	
ABCDEF	AE	ABCDEF-001	AESEQ	2	SUPPVAR1	[label]	N	CRF	
ABCDEF	AE	ABCDEF-001	AESEQ	2	SUPPVAR2	[label]	2012-02-09	CRF	

In this project the ‘target metadata’ was defined using a relatively simple approach that combined the information from the CDISC eShare Excel spreadsheets with the information from Excel spreadsheets containing company additions. Part of the code within the %makefinaldomain macro is greatly depending on the way the ‘target metadata’ is defined. However, because of the modular structure of the framework, it is relatively easy to adapt or replace the %makefinaldomain macro in order to make it work with other (structured) metadata files (e.g. your current company standard). It has no impact on other parts of this framework.

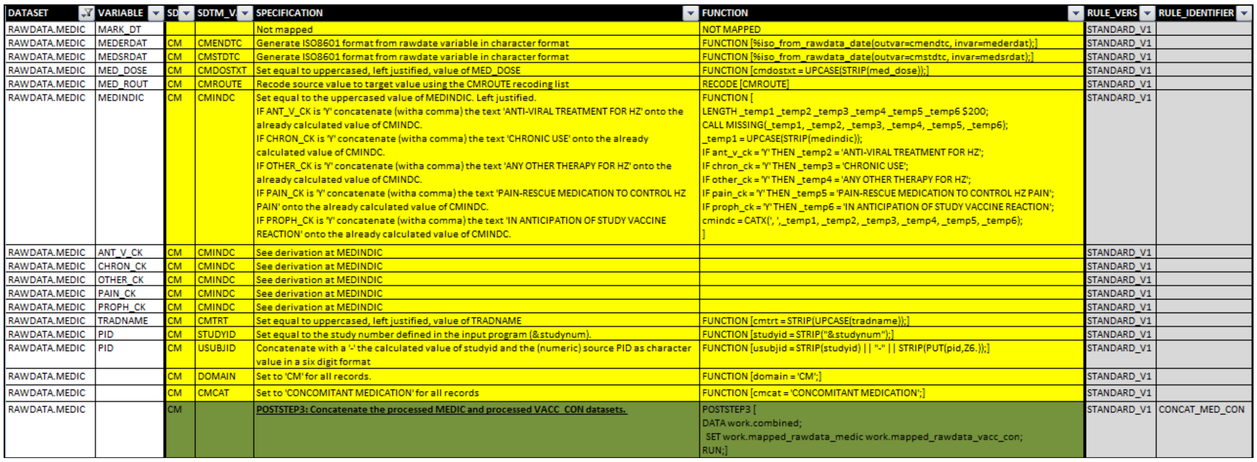
THE USE OF STANDARD MAPPINGS

The mapping specification document is a relatively big document containing not only all source variables and all target variables, but also the mapping specifications and (pseudo-)code needed to generate the targets from the sources. This document can become relatively large and generating it from scratch for each new trial is labour-intensive and therefore needs to be prevented.

Fortunately, these days most companies are working in a relatively standard environment. If the sources are standardized it is possible to specify **standard** mappings rules which enable to go from a **standard** source via a **standard** mapping to a **standard** target. On the other hand it should still be possible to deviate from a standard and specify a non-standard mapping rule (e.g. for a non-standard source variable or a standard source variable that is to be handled in a non-standard way).

In order to facilitate both (standard and non-standard) options, an approach has been implemented that makes use of a standard mapping specification document in which the standard mapping rules are specified. This document will have the same structure as the mapping specification document as is shown in the picture on the following page:

Standard mapping specification document:



DATASET	VARIABLE	SD	SDTM_V	SPECIFICATION	FUNCTION	RULE_VERS	RULE_IDENTIFIER
RAWDATA.MEDIC	MARK_DT			Not mapped	NOT MAPPED	STANDARD_V1	
RAWDATA.MEDIC	MEDERDAT	CM	CMENDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=cmendtc, invar=mederdat);]	STANDARD_V1	
RAWDATA.MEDIC	MEDSRDAT	CM	CMSTDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=cmstdtc, invar=medsrdat);]	STANDARD_V1	
RAWDATA.MEDIC	MED_DOSE	CM	CMDOSTXT	Set equal to uppercased, left justified, value of MED_DOSE	FUNCTION [cmdostxt = UPCASE(STRIP(med_dose));]	STANDARD_V1	
RAWDATA.MEDIC	MED_ROUT	CM	CMROUTE	Recode source value to target value using the CMROUTE recoding list	RECODE [CMROUTE]	STANDARD_V1	

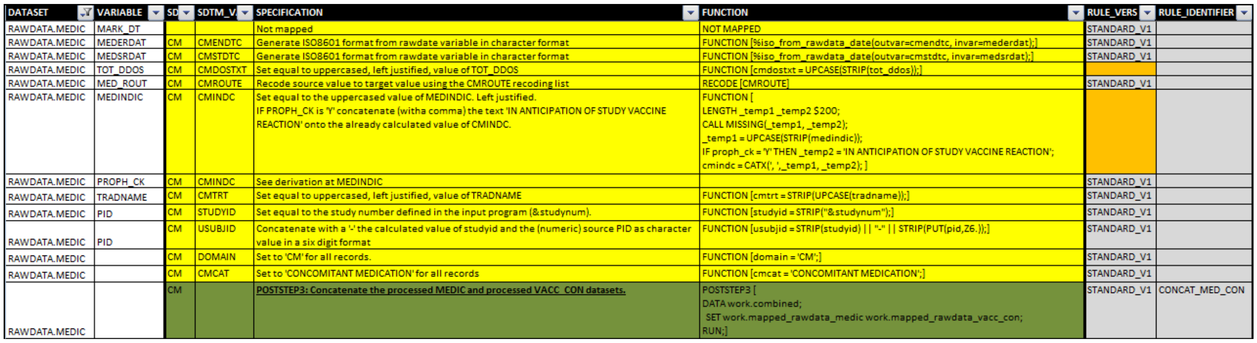
RAWDATA.MEDIC	MEDINDIC	CM	CMINDC	Set equal to the uppercased value of MEDINDIC. Left justified.
				IF ANT_V_CK is 'Y' concatenate (witha comma) the text 'ANTI-VIRAL TREATMENT FOR HZ' onto the already calculated value of CMINDC.
				IF CHRON_CK is 'Y' concatenate (witha comma) the text 'CHRONIC USE' onto the already calculated value of CMINDC.
				IF OTHER_CK is 'Y' concatenate (witha comma) the text 'ANY OTHER THERAPY FOR HZ' onto the already calculated value of CMINDC.
				IF PAIN_CK is 'Y' concatenate (witha comma) the text 'PAIN-RESCUE MEDICATION TO CONTROL HZ PAIN' onto the already calculated value of CMINDC.
				IF PROPH_CK is 'Y' concatenate (witha comma) the text 'IN ANTICIPATION OF STUDY VACCINE REACTION' onto the already calculated value of CMINDC.
					FUNCTION [
					LENGTH _temp1 _temp2 _temp3 _temp4 _temp5 _temp6 $200;
					CALL MISSING(_temp1, _temp2, _temp3, _temp4, _temp5, _temp6);
					_temp1 = UPCASE(STRIP(medindic));
					IF ant_v_ck = 'Y' THEN _temp2 = 'ANTI-VIRAL TREATMENT FOR HZ';
					IF chron_ck = 'Y' THEN _temp3 = 'CHRONIC USE';
					IF other_ck = 'Y' THEN _temp4 = 'ANY OTHER THERAPY FOR HZ';
					IF pain_ck = 'Y' THEN _temp5 = 'PAIN-RESCUE MEDICATION TO CONTROL HZ PAIN';
					IF proph_ck = 'Y' THEN _temp6 = 'IN ANTICIPATION OF STUDY VACCINE REACTION';
					cmindc = CATX(', ',_temp1, _temp2, _temp3, _temp4, _temp5, _temp6);
					]
						STANDARD_V1	

RAWDATA.MEDIC	ANT_V_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	CHRON_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	OTHER_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	PAIN_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	PROPH_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	TRADNAME	CM	CMTRT	Set equal to uppercased, left justified, value of TRADNAME	FUNCTION [cmtrt = STRIP(UPCASE(tradname));]	STANDARD_V1	
RAWDATA.MEDIC	PID	CM	STUDYID	Set equal to the study number defined in the input program (&studynum).	FUNCTION [studyid = STRIP("&studynum");]	STANDARD_V1	
RAWDATA.MEDIC	PID	CM	USUBJID	Concatenate with a '-' the calculated value of studyid and the (numeric) source PID as character value in a six digit format	FUNCTION [usubjid = STRIP(studyid) | | "-" | | STRIP(PUT(pid,Z6.));]	STANDARD_V1	
RAWDATA.MEDIC		CM	DOMAIN	Set to 'CM' for all records.	FUNCTION [domain = 'CM';]	STANDARD_V1	
RAWDATA.MEDIC		CM	CMCAT	Set to 'CONCOMITANT MEDICATION' for all records	FUNCTION [cmcat = 'CONCOMITANT MEDICATION';]	STANDARD_V1	

RAWDATA.MEDIC		CM		POSTSTEP3: Concatenate the processed MEDIC and processed VACC_CON datasets.	POSTSTEP3 [
					DATA work.combined;
					 SET work.mapped_rawdata_medic work.mapped_rawdata_vacc_con;
					RUN;]
						STANDARD_V1	CONCAT_MED_CON

In the mapping specification document of the **trial** the RULE_VERS and RULE_IDENTIFIER columns are added that can be used to point to a standard mapping in the standard mapping specification document. Or, alternatively, these columns can be left empty to indicate the source-to-target mapping is a trial specific, non-standard mapping (cells are orange because of the conditional formatting):

Trial mapping specification document:



DATASET	VARIABLE	SD	SDTM_V	SPECIFICATION	FUNCTION	RULE_VERS	RULE_IDENTIFIER
RAWDATA.MEDIC	MARK_DT			Not mapped	NOT MAPPED	STANDARD_V1	
RAWDATA.MEDIC	MEDERDAT	CM	CMENDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=cmendtc, invar=mederdat);]	STANDARD_V1	
RAWDATA.MEDIC	MEDSRDAT	CM	CMSTDTC	Generate ISO8601 format from rawdate variable in character format	FUNCTION [%iso_from_rawdata_date(outvar=cmstdtc, invar=medsrdat);]	STANDARD_V1	
RAWDATA.MEDIC	TOT_DDOS	CM	CMDOSTXT	Set equal to uppercased, left justified, value of TOT_DDOS	FUNCTION [cmdostxt = UPCASE(STRIP(tot_ddos));]		
RAWDATA.MEDIC	MED_ROUT	CM	CMROUTE	Recode source value to target value using the CMROUTE recoding list	RECODE [CMROUTE]	STANDARD_V1	

RAWDATA.MEDIC	MEDINDIC	CM	CMINDC	Set equal to the uppercased value of MEDINDIC. Left justified.
				IF PROPH_CK is 'Y' concatenate (witha comma) the text 'IN ANTICIPATION OF STUDY VACCINE REACTION' onto the already calculated value of CMINDC.
					FUNCTION [
					LENGTH _temp1 _temp2 $200;
					CALL MISSING(_temp1, _temp2);
					_temp1 = UPCASE(STRIP(medindic));
					IF proph_ck = 'Y' THEN _temp2 = 'IN ANTICIPATION OF STUDY VACCINE REACTION';
					cmindc = CATX(', ',_temp1, _temp2);]		

RAWDATA.MEDIC	PROPH_CK	CM	CMINDC	See derivation at MEDINDIC		STANDARD_V1	
RAWDATA.MEDIC	TRADNAME	CM	CMTRT	Set equal to uppercased, left justified, value of TRADNAME	FUNCTION [cmtrt = STRIP(UPCASE(tradname));]	STANDARD_V1	
RAWDATA.MEDIC	PID	CM	STUDYID	Set equal to the study number defined in the input program (&studynum).	FUNCTION [studyid = STRIP("&studynum");]	STANDARD_V1	
RAWDATA.MEDIC	PID	CM	USUBJID	Concatenate with a '-' the calculated value of studyid and the (numeric) source PID as character value in a six digit format	FUNCTION [usubjid = STRIP(studyid) | | "-" | | STRIP(PUT(pid,Z6.));]	STANDARD_V1	
RAWDATA.MEDIC		CM	DOMAIN	Set to 'CM' for all records.	FUNCTION [domain = 'CM';]	STANDARD_V1	
RAWDATA.MEDIC		CM	CMCAT	Set to 'CONCOMITANT MEDICATION' for all records	FUNCTION [cmcat = 'CONCOMITANT MEDICATION';]	STANDARD_V1	

RAWDATA.MEDIC		CM		POSTSTEP3: Concatenate the processed MEDIC and processed VACC_CON datasets.	POSTSTEP3 [
					DATA work.combined;
					 SET work.mapped_rawdata_medic work.mapped_rawdata_vacc_con;
					RUN;]
						STANDARD_V1	CONCAT_MED_CON

In the example of the trial above, standard rules are applied for nearly all source to target mappings except for the creation of the CMDOSTXT variable (a different source variable is used) and the creation of the CMINDC variable (there are fewer source variables used to generate the CMINDC target variable).

Please note that if a standard rule is referenced in the trial mapping specification document (indicated by the column RULE_VERS), the rule as defined in the standard will be obtained and will be used by the %mapping and %apply_poststeps macros. The specification and (pseudo-)code are still available in the trial mapping specification document, but these are copies and kept only for clarity reasons. A standard program is generated that will compare the specification and function specified in the trial mapping specification document with the rules in the standard mapping specification document and will give warning messages if differences are encountered. It will also give a message if a standard rule is not found. The program will use the first four columns (with the source and target datasets/variables) and the RULE_VERS and RULE_IDENTIFIER columns from the trial mapping specification document as the key variables to obtain the standard specification and (pseudo-) code from the standard mapping specification document. By using different values for RULE_VERS it is possible to store multiple versions of mapping rules in the standard mapping specification document. This makes it possible to define different standard rules for the same combination of source and target datasets/variables.

Using the standard mapping specification document an initial version of the mapping specification document can be generated for a trial, which will replace the majority of the labour-intensive work of generating the mapping specification document from scratch. A standard program is generated that facilitates this; it uses the standard mapping specification document and the structure of the source data of a trial to generate the trial mapping specification document with the latest version of all source-to-target mappings for the existing source variables. The resulting initial mapping specification document needs only to be adapted for study specific additions or study specific changes.

THE BENEFITS

The mapping framework described in this paper is easy to understand and, because of its modular structure, the framework can be implemented with minimum effort.

One of the major benefits of the mapping framework is the readability of the source-to-target mapping specifications:

- All source-to-target mapping specifications are available in one Excel sheet;
- The human-readable source-to-target specification and (pseudo-)code are shown directly next to each other;
- The Excel format allows easy filtering and impact assessment for any changes to the source and/or target;
- There is a close link between the source data, the mapping rules and the output data which ensures that the mapping specialist is ‘*close to the data*’. This results in a more reliable and trustworthy conversion process.

The use of a separate standard mapping specification document is also considered to be very beneficial:

- Standard and non-standard mappings can easily be identified, making it easy to apply a different quality control/validation process;
- There is no need to re-specify standard mappings, a reference to a standard mapping specification can be made instead;
- A program is made available which compares the source data of a trial with the standard mapping specification and automatically obtains the standard mappings for all known source variables. This can be used as basis of the trial specific mapping specification document;

A few other benefits of the mapping approach:

- Standard metadata is used on several locations in the process which ensures all output generated is according to the defined standard. It is not possible to generate output that is not defined in the standards;
- All code that is generated and executed is SAS code and the MFILE option is used to print all executed code to a file. This is particularly useful when debugging or when the code needs to be sent to external parties;
- Warnings/errors are generated when the mapping specification document does not align with the source data or the standard mapping specification document;
- The available functions as currently available in the mapping specification document (in the column “FUNCTION”) can be extended if the need arises. For example, the STACK function was added at a later time.

CONTACT INFORMATION

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

Author Name Bas van Bakel
Company OCS Life Sciences
Address Ruwekampweg 2G
City / Postcode ‘s-Hertogenbosch / 5222 AT
Work Phone: +31 (0)73 523 6000
Email: info.nl@ocs-consulting.com
Web: www.ocs-lifesciences.com

Brand and product names are trademarks of their respective companies.