Paper SI05

Standardization = Automation: A Journey from Dataset Specification to SAS Code

Anastasiia Khmelnytska, Intego Group LLC, Kharkiv, Ukraine

ABSTRACT

The beauty of standards is in the fact that well-defined structure screams automation. Creating datasets according to CDISC guidelines has never been easier with the macro that transforms your dataset specification into SAS® code. It will make you forget about manually defining variables' attributes and parameters once and for all. The step by step guide will also cover the mappings of standard variables and parameters defined in the specification. In this paper we will discuss several approaches to creation of dataset specifications that can be used effectively for automatic code generation. Whether you are creating SDTM datasets from raw data, or mapping SDTM data to ADaM standards, with this advanced technique, you will be able to create different datasets with a single macro call – all you have to do is to choose your parameters wisely.

INTRODUCTION

One of my university professors once said that mathematicians are lazy – that's why they like to come up with short and elegant solutions when possible. I guess you can say the same about programmers. Time is a valuable asset and no one likes to spend it on monotonous repetitive tasks. Besides, in our industry time is also critical to the patients: the sooner we complete a clinical trial the sooner our patients get the treatment they need. That's why one of the main trends in clinical research is shorter timelines, faster filings, and hence the growing need for automation.

Data standards help FDA and other regulatory authorities to review submissions more efficiently and to deliver treatments to patients faster. However, they also allow us programmers to implement these standards by developing various automation techniques: this way we make our codes interoperable the same way our data already is. Also, the more automated data transformation process is – the less are the chances of human error.

In this paper we will discuss some of the most common programming tasks that can be easily automated using macros. The first part of the paper will cover the %attrib macro that allows you to create dataset and variable metadata according to CDISC standards directly from the specification. It is convenient not only for the initial creation of a dataset, but even more so for when you need to update some of the metadata: once the specification is updated, so is your dataset. The second part will cover %formats macro that creates formats from the EXCEL spreadsheet based on the controlled terminology defined. This is one of the approaches to specification design that gives you more room for automation. Another one is value-level metadata – we will discuss what it is, what the benefits are and when it can be used. It will be covered in the third part of the paper which introduces %vlm macro that transforms "Value-Level Metadata" spreadsheet of the specification into SAS code.

%ATTRIB MACRO

The first thing that comes to mind when you think about automation based on dataset specifications is reading in dataset metadata. %attrib macro can help you with that:

```
%macro attrib (type   = , /*Data type: SDTM or ADAM*/
               spec   = , /*Location of specifications file*/
               domain = , /*Name of SDTM or ADAM dataset that you need metadata for*/
               dsnin  = , /*Name of input dataset*/
               outlib =   /*Libname for the location where final dataset will be
stored*/  ) / minoperator;
```

So what exactly can you do with this macro?

SOME SIMPLE CHECKS

First of all, depending on the dataset type that you've stated we can check whether the domain name provided is a valid SDTM domain or ADAM dataset name according to CDISC standards:

```
%if &type. = SDTM %then %do;
  %if not (&domain. in AE AG APAE APBS APDD APDM APEX APLB APMH APQS APRELSUB APRP
                       APSC APSS APSU BE BS CE CM CO CV DA DD DE DI DM DO DR DS DT DU
                       DV DX EC ED EG EX FA FT GI HM HO IE IS LB MB MH MI MK ML MO MS
                       NV OE OI PB PC PE PF PG POOLDEF PP PR QS RE RELREC RELSUB RP RS
                       SB SC SE SM SR SS SU SUPPAE SUPPAG SUPPBE SUPPBS SUPPCE SUPPCM
                       SUPPCV SUPPDA SUPPDD SUPPDE SUPPDM SUPPDO SUPPDS SUPPDT SUPPDU
                       SUPPDV SUPPDX SUPPEC SUPPEG SUPPER SUPPEX SUPPFA SUPPHO SUPPIE
                       SUPPIS SUPPLB SUPPMB SUPPMH SUPPMI SUPPMK SUPPML SUPPMO SUPPMS
```

```
                        SUPPNV SUPPOE SUPPPB SUPPPC SUPPPE SUPPPF SUPPPG SUPPPP SUPPPR
                        SUPPQS SUPPQT SUPPRE SUPPRP SUPPRS SUPPSB SUPPSC SUPPSM SUPPSR
                        SUPPSS SUPPSU SUPPTR SUPPTU SUPPUR SUPPVS SUPPWB SV TA TD TE TI
                        TM TR TS TU TV TX UR VS) %then %do;
    %put %str(WAR)NING: &domain. is not a valid SDTM domain name.;
    %return;
  %end;
%end;

%else %if &type. = ADAM and (%substr(&domain., 1, 2) ^= AD %then %do;
    %put %str(WAR)NING: &domain. is not a valid ADAM dataset name.;
    %return;
%end;
```

If the domain name provided is not on the list of CDISC standard SDTM domains or if ADAM dataset name does not start with "AD" then a warning message will be printed to the log and the macro will stop executing. If you want the macro to execute anyway, you can simply remove the %return statement from the code above.

VARIABLES' ATTRIBUTES

After we have established that all is good with domain name, we can start by getting the list of all variables in the dataset, their type, length and label directly from specification. Let's use vital signs SDTM.VS dataset as an example. Here is the metadata that we have in our EXCEL spreadsheet for this dataset:

1	Variable Name	Variable Label	Type	Core	Length
2	STUDYID	Study Identifier	Char	Req	13
3	DOMAIN	Domain Abbreviation	Char	Req	2
4	USUBJID	Unique Subject Identifier	Char	Req	24
5	VSSEQ	Sequence Number	Num	Req	8
6	VSTESTCD	Vital Signs Test Short Name	Char	Req	8
7	VSTEST	Vital Signs Test Name	Char	Req	40
8	VSORRES	Result or Finding in Original Units	Char	Exp	8
9	VSORRESU	Original Units	Char	Exp	11
10	VSSTRESC	Character Result/Finding in Std Format	Char	Exp	8
11	VSSTRESN	Numeric Result/Finding in Standard Units	Num	Exp	8
12	VSSTRESU	Standard Units	Char	Exp	11
13	VSSTAT	Completion Status	Char	Perm	8
14	VSBLFL	Baseline Flag	Char	Exp	1
15	VISITNUM	Visit Number	Num	Exp	8
16	VISIT	Visit Name	Char	Perm	8
17	VSDTC	Date/Time of Measurements	Char	Exp	19
18	VSDY	Study Day of Vital Signs	Num	Perm	8
19	VSTPT	Planned Time Point Name	Char	Perm	13
20	VSTPTNUM	Planned Time Point Number	Num	Perm	8

Picture 1. VS dataset specifications

Using PROC IMPORT we can easily convert this EXCEL spreadsheet to SAS dataset:

```
proc import datafile = &spec.
            out      = &domain.attrib (keep = variable_name variable_label type length)
            dbms     = xlsx
            replace;
   sheet = "&domain.";
run;
```

The parameters that you need to set in PROC IMPORT are:

1. datafile – your input EXCEL file with specification
2. out – output dataset name

3. dbms – type of data that you import (xlsx for EXCEL in our case)
4. replace – if the dataset with such name already exists, it is replaced

You also need to specify a SHEET statement with the name of EXCEL sheet for the domain that you are currently working on. This is where macro variable "domain" (which is one of the parameters of the %attrib macro) can be used.

Note that by default SAS gets variable names from the first row of your spreadsheet (if you don't want that, you should state option GETNAMES = NO), however if you have any spaces in your column names they will be transformed to underscore in the name of the variable. You will get a note in log about this:

```
NOTE:    Variable Name Change.  Variable Name -> Variable_Name
NOTE:    Variable Name Change.  Variable Label -> Variable_Label
```

Now that we have all the necessary information stored in a dataset VS_ATTRIB, we can create some macro variables for further use:

```
%let VAR    =        /*List of all variables in the dataset*/;
%let ATTRIB = attrib /*The ATTRIB statement with variables' lengths and labels*/;

data _null_;
   set &domain._attrib;
   call symput ("VAR", symget("VAR")||" "||strip(variable_name));
   if type = "Char" then stype = "$";
   call symput ("ATTRIB", symget("ATTRIB")||""||strip(variable_name)||" label='"||
                strip(variable_label)||"' length="||stype||strip(put(length, 3.)));

run;
```

In the DATA _NULL_ step above we assign values to two macro variables VAR and ATTRIB using CALL SYMPUT.

Macro variable VAR is initialized as missing, and with each iteration of the data step new variable name is appended to the list from the previous step. This way, at the end of data step execution macro variable VAR will contain the list of all variables that we want to keep in our final dataset. Note that we use symget("VAR") rather than &VAR. that is more commonly used to reference a macro variable. That is because &VAR. resolves during macro run time, before the data step execution begins, meaning that it resolves only once. On the other hand, symget("VAR") resolves during SAS run time which makes it possible to resolve to different values at each iteration of the data step – exactly what we need here. Same reason for using CALL SYMPUT rather then %let macro statement.

Macro variable ATTRIB will contain the attrib statement that will be used to assign variables' lengths and labels. The statement's syntax is as follows:

attrib var1 var-attributes1
…
varN var-attributesN;

where *var-attributesX* may include FORMAT=format, INFORMAT=informat, LABEL='label' and LENGTH=<$>length. We initialize variable ATTRIB with "attrib" (the statement itself) and then add variable name concatenated with its label and length at each iteration of the data step for each variable. We also create an intermediate variable STYPE that either contains a $ sign or is missing depending on variable type (character or numeric) – it is used while parsing the LENGTH variable option.

After this step is executed macro variables have the following values:

Variable Name	Variable Value
VAR	STUDYID DOMAIN USUBJID VSSEQ VSTESTCD VSTEST VSORRES VSORRESU VSSTRESC VSSTRESN VSSTRESU VSSTAT VSBLFL VISITNUM VISIT VSDTC VSDY VSTPT VSTPTNUM
ATTRIB*	attrib STUDYID label='Study Identifier' length=$13 DOMAIN label='Domain Abbreviation' length=$2 USUBJID label='Unique Subject Identifier' length=$24 VSSEQ label='Sequence Number' length= 8 VSTESTCD label='Vital Signs Test Short Name' length=$8 VSTEST label='Vital Signs Test Name' length=$40 VSORRES label='Result or Finding in Original Units' length=$8 VSORRESU label='Original Units' length=$11 VSSTRESC label='Character Result/Finding in Std Format' length=$8 VSSTRESN label='Numeric Result/Finding in Standard Units' length= 8 VSSTRESU label='Standard Units' length=$11 VSSTAT label='Completion Status' length=$8 VSBLFL label='Baseline Flag' length=$1 VISITNUM label='Visit Number' length= 8 VISIT label='Visit Name' length=$8 VSDTC label='Date/Time of Measurements' length=$19

	VSDY label='Study Day of Vital Signs' length= 8 VSTPT label='Planned Time Point Name' length=$13 VSTPTNUM label='Planned Time Point Number' length= 8

Table 1. Values of macro variable VAR and ATTRIB

*macro variable ATTRIB will not have this exact indenting, this was done in the paper for visual purposes only.

DATASET METADATA

Another thing that you can do with the %attrib macro is get dataset metadata such as dataset label and unique keys from the specification. You will normally have a sheet called "Dataset Metadata" in your EXCEL file with the structure similar to the example below:

1	Dataset	Description	Class	Structure	Keys
18	VS	Vital Signs	Findings	One record per vital sign measurement per time point per visit per subject	STUDYID, USUBJID, VSTESTCD, VISITNUM, VSTPTNUM

Picture 2. VS dataset metadata

In the similar way as before, we use PROC IMPORT to read in Dataset Metadata sheet from EXCEL file to SAS dataset, and then create macro variables KEYS and LABEL to store the unique key and dataset label.

```
proc import datafile = S_Specs
            out      = &domain._sort (keep = dataset description keys)
            dbms     = xlsx
            replace;
   sheet = "Dataset Metadata";
run;

data _null_;
   set &domain._sort;
   where dataset = "&domain.";
   call symput("keys",  compress(keys, ","));
   call symput("label", strip(description));

run;
```

Now that we have all the information we need, we can output the final dataset to &outlib. library (which is one of the parameters of the macro):

```
data &outlib..&domain. (label = "&label" keep = &var.);
   &attrib.;
   set &dsnin.;
   %if &type. = SDTM %then %do;
     format _all_;
   %end;
run;

proc sort data = &outlib..&domain.;
   by &keys.;
run;
```

In the data step above we create the final dataset from our input dataset defined by DSNIN parameter of the macro, set all the attributes, keep only necessary variables, assign dataset label and sort the dataset by its unique key.

Call this macro at the end of your program and you are good to go:

```
%attrib(type   = SDTM,
        spec   = "your-location\SDTM_Specs.xlsx",
        domain = VS,
        dsnin  = __vs);
```

%FORMATS MACRO

Another thing that is easy to automate is deriving variables that only need formatting compared to the original value. Examples may include mapping –TESTCD, –TEST, --METHOD, etc. from raw data, creating VISIT and VISITNUM, deriving PARAM based on SDTM variables and so on. Some of these variables will have codelists provided by CDISC, others will not, so you will have to create your own terminology for them. You can add a separate spreadsheet called "Controlled Terminology" in your specification that will contain both. Now, using the %formats macro you can easily create formats for all of these variables from dataset specification without the need to define them manually in the program:

```
%macro formats(domain = ,/*Name of the domain for which you want to create formats*/
               spec   =  /*Location of specifications file*/);
```

Here is the example of “Controlled Terminology” spreadsheet with codelists for Vital Signs domain, as well as for some variables that can be used across multiple domains (i.e. VISIT):

Domain	Codelist	Reported value	Submission value
MULTIPLE	VISIT	SCRNBASE	Screening
MULTIPLE	VISIT	DAY1	Day 1
MULTIPLE	VISIT	DAY2	Day 2
MULTIPLE	VISIT	DAY3	Day 3
MULTIPLE	VISIT	DAY4	Day 4
MULTIPLE	VISIT	DAY5	Day 5
MULTIPLE	VISIT	DAY6	Day 6
MULTIPLE	VISIT	DAY7	Day 7
MULTIPLE	VISIT	DAY8	Day 8
MULTIPLE	VISIT	DAY9	Day 9
MULTIPLE	VISIT	DAY10	Day 10

Domain	Codelist	Reported value	Submission value
MULTIPLE	VISITNUM	SCRNBASE	-30
MULTIPLE	VISITNUM	DAY1	1
MULTIPLE	VISITNUM	DAY2	2
MULTIPLE	VISITNUM	DAY3	3
MULTIPLE	VISITNUM	DAY4	4
MULTIPLE	VISITNUM	DAY5	5
MULTIPLE	VISITNUM	DAY6	6
MULTIPLE	VISITNUM	DAY7	7
MULTIPLE	VISITNUM	DAY8	8
MULTIPLE	VISITNUM	DAY9	9
MULTIPLE	VISITNUM	DAY10	10

Domain	Codelist	Reported value	Submission value
VS	VSTESTCD	VSBPDIA	DIABP
VS	VSTESTCD	HEIGHT	HEIGHT
VS	VSTESTCD	VSPULSE	PULSE
VS	VSTESTCD	VSRESP	RESP
VS	VSTESTCD	VSBPSYS	SYSBP
VS	VSTESTCD	VSTEMP	TEMP
VS	VSTESTCD	WEIGHT	WEIGHT

Domain	Codelist	Reported value	Submission value
VS	VSTEST	DIABP	Diastolic Blood Pressure
VS	VSTEST	HEIGHT	Height
VS	VSTEST	PULSE	Pulse Rate
VS	VSTEST	RESP	Respiratory Rate
VS	VSTEST	SYSBP	Systolic Blood Pressure
VS	VSTEST	TEMP	Temperature
VS	VSTEST	WEIGHT	Weight

Domain	Codelist	Reported value	Submission value
VS	VSSTRESU	DIABP	mmHg
VS	VSSTRESU	HEIGHT	cm
VS	VSSTRESU	PULSE	BEATS/MIN
VS	VSSTRESU	RESP	BREATHS/MIN
VS	VSSTRESU	SYSBP	mmHg
VS	VSSTRESU	TEMP	C
VS	VSSTRESU	WEIGHT	kg

Domain	Codelist	Reported value	Submission value
VS	VSTPT	0	PREDOSE
VS	VSTPT	1	1 HOUR POSTDOSE
VS	VSTPT	2	2 HOURS POSTDOSE
VS	VSTPT	3	3 HOURS POSTDOSE
VS	VSTPT	4	4 HOURS POSTDOSE
VS	VSTPT	6	6 HOURS POSTDOSE
VS	VSTPT	8	8 HOURS POSTDOSE

Picture 2. Controlled terminology example for VS dataset

Domain column can either contain an actual name of the domain that the particular format corresponds to, or it can contain the word “MULTIPLE” in case it can be used across various domains. Also, if there can be several reported values that need to be mapped into one submission value, you can list them all in one row separated by “~” sign instead of creating multiple rows.

Same as with %attrib macro, first thing that you need to do is import the spreadsheet with controlled terminology into SAS dataset. We keep only those records from the spreadsheet that are needed for our domain (in our case, with DOMAIN = VS or DOMAIN = MULTIPLE):

```
proc import datafile = &spec.
            out      = fmt (rename = (reported_value    = repvalue
                                      submission_value = ctvalue))
            dbms     = xlsx
            replace;
   sheet = "Controlled Terminology";
run;

proc sort data = fmt;
   by domain codelist repvalue ctvalue;
   where domain in ("&domain.", "MULTIPLE");
run;
```

Now we need to create a dataset that can be used as an input dataset for PROC FORMAT with CNTLIN option. In order for that to be possible, this dataset should have specific structure. There are three required variables:

- FMTNAME – name of the format
- START – variable that contains the “from” value (left of “=” sign in PROC FORMAT)
- LABEL – variable that contains the “to” value (right of “=” sign in PROC FORMAT)

Another variable that is not mandatory, but also very important is TYPE. This variable determines whether you are creating a format or informat and whether it is numeric or character. There are five possible values of TYPE variable:

Value	Stands for	Compatible with	Converts from	Converts to
C	Character format	PUT function	Character	Character
N	Numeric format	PUT function	Numeric	Character
J	Character informat	INPUT function	Character	Character
I	Numeric informat	INPUT function	Character	Numeric
P	Picture format	PUT function	Numeric	Character

Table 2. Types of formats and informats

Picture format is a specific kind of format and it is not suitable for our current needs. Character format and character informat both convert variables from character to character, so this is just a question of whether you prefer to use

PUT or INPUT. I prefer PUT function, so I am keeping three possible values of TYPE variable: C, N and I. If you don't create this variable in your dataset, SAS will automatically assume that TYPE is "C" if FMTNAME start with a $ and "N" if it doesn't. As we don't plan on having any $ signs in our spreadsheet, we will have to create TYPE variable explicitly in our program.

As a first step, we will need to determine whether our reported value and submission value are character or numeric:

```
data fmt_type;
  set fmt;
  by domain codelist repvalue ctvalue;
  length type_rep type_ct $4;
  retain type_rep type_ct;
  if first.codelist then do;
    type_rep = "num";
    type_ct  = "num";
  end;
  if verify(strip(repvalue), '0123456789-.~') then type_rep = "char";
  if verify(strip(ctvalue),  '0123456789-.~') then type_ct  = "char";
  if last.codelist;
  drop repvalue ctvalue;
run;
```

First we will assume for each format that both REPVALUE and CTVAUE are numeric and then if we find at least one value that is not numeric (VERIFY function returns a position of the character that is not in '0123456789-.~' list) we say that the type is character. This way we will have a type of both reported value and submission value associated with each format name. In the example above, we will have the following FMT_TYPE dataset for our Vital Signs formats:

	Domain	Codelist	type_rep	type_ct
1	MULTIPLE	VISIT	char	char
2	MULTIPLE	VISITNUM	char	num
3	VS	VSSTRESU	char	char
4	VS	VSTEST	char	char
5	VS	VSTESTCD	char	char
6	VS	VSTPT	num	char

Picture 3. Intermediate FMT_TYPE dataset for VS domain

Now, based on the table above, we assign the TYPE variable for each format to C, N or I:

```
data fmt2;
  merge fmt fmt_type;
  by domain codelist;
  retain hlo "";
       if type_rep = "char" and type_ct = "char" then type = "c";
  else if type_rep = "char" and type_ct = "num"  then type = "i";
  else if type_rep = "num"  and type_ct = "char" then type = "n";
  if index(repvalue, "~") then do;
      repvalue = repvalue;
    do i = 1 to countw(__repvalue, "~");
      repvalue = scan(__repvalue, i, "~");
      output;
    end;
  end;
  else output;
  drop domain type_rep type_ct __repvalue i;
run;
```

Here we also create a new record for each reported value in case there were several of them listed in one using "~" sign.

Now the only thing left to do is to create the formats using PROC FORMAT with CNTLIN option and rename the original variables to the names that PROC FORMAT expects:

```
proc format cntlin = fmt2 (rename = (codelist = fmtname
                                     repvalue = start
                                     ctvalue  = label));
run;
```

This way in your dataset program you will have to call the %formats macro with the correct parameters and just apply the formats it creates:

```
%formats(domain = VS,
         spec   = "your-location\SDTM_Specs.xlsx");
```

```
data vs;
  set raw.vs;
  vstestcd = put(vsnam, $vstestcd.);
  vstest   = put(vsnam, $vstest.);
  vsstresu = put(vsnam, $vsstresu.);
  visit    = put(visid, $visit.);
  visitnum = input(visid, visitnum.);
  vstpt    = put(tpt, vstpt.);
run;
```

%VLM MACRO

Dataset specification in its standard form is designed to describe each variable's metadata and its derivation. However sometimes, more often in ADaM than in SDTM, some variables' derivations depend on the values of other variables. For example, PARAM can depend on the values of —TESTCD, —CAT, —METHOD, etc. or the value of AVAL may be derived differently for each PARAMCD. That's when it is convenient to use value-level metadata in the specification instead of variable-level. This way you make your specs more readable and easier to understand. Another advantage is that with value-level metadata it is easy to automate derivations of variables defined in such way. Let's consider some examples from "Value Level Metadata" sheet in ADaM specification for ADLB dataset:

DATASET	VARIABLE	WHERE	ALGORITHM	CTVALUE	DATATYPE	LENGTH
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='ALB'	Set to value in CTVALUE	Albumin (g/dL)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='ALP'	Set to value in CTVALUE	Alkaline Phosphatase (U/L)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='ALT'	Set to value in CTVALUE	Alanine Aminotransferase, ALT (U/L)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='AST'	Set to value in CTVALUE	Aspartate Aminotransferase, AST (U/L)	text	40
ADLB	PARAM	LBCAT='HEMATOLOGY' and LBTESTCD='BASO'	Set to value in CTVALUE	Basophils (10^9/L)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='BILI'	Set to value in CTVALUE	Total Bilirubin (mg/dL)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='CA'	Set to value in CTVALUE	Total Calcium (mg/dL)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='CHOL'	Set to value in CTVALUE	Total Cholesterol (mg/dL)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='CL'	Set to value in CTVALUE	Chloride (mmol/L)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='CREAT'	Set to value in CTVALUE	Creatinine (mg/dL)	text	40
ADLB	PARAM	LBCAT='HEMATOLOGY' and LBTESTCD='EOS'	Set to value in CTVALUE	Eosinophils (10^9/L)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='GLUC'	Set to value in CTVALUE	Glucose (mg/dL)	text	40
ADLB	PARAM	LBCAT='HEMATOLOGY' and LBTESTCD='HCT'	Set to value in CTVALUE	Hematocrit (%)	text	40
ADLB	PARAM	LBCAT='HEMATOLOGY' and LBTESTCD='HGB'	Set to value in CTVALUE	Hemoglobin (g/dL)	text	40
ADLB	PARAM	LBCAT='CHEMISTRY' and LBTESTCD='K'	Set to value in CTVALUE	Potassium (mmol/L)	text	40
ADLB	PARAM	LBCAT='HEMATOLOGY' and LBTESTCD='LYM'	Set to value in CTVALUE	Lymphocytes (10^9/L)	text	40

Picture 4. Value-level metadata example for ADLB.PARAM

We can see that variable PARAM has different values based on the corresponding values of LBCAT and LBTESTCD. Using %vlm macro we can read in this value-level metadata and automatically generate the corresponding if-then conditional lines of code.

```
%macro vlm (domain = ,/*Name of the domain for which you want to create if-then
                       statements from value-level metadata*/
            spec   =  /*Location of specifications file*/);
```

First step, as it was before, is to import the spreadsheet:

```
proc import datafile = &spec.
            out      = vlm
            dbms     = xlsx
            replace;
   sheet = "Value Level Metadata";
run;
```

Now we need to define the macro variable VLM that will hold all of the statements that we will generate. We need to make sure that it is a global macro variable, so that it would exist outside of our macro in our dataset program code, and also we set it to missing so that it doesn't contain any values from previous runs. After that we can start processing our VLM dataset created from value-level metadata:

```
%global vlm;
%let vlm = ;

data _null_;
  set vlm;
  where dataset="&domain.";
  first-block-of-code
  second-block-of-code
  third-block-of-code
run;
```

We will create if-then statements and put them into a VLM macro variable within a DATA _NULL_ step. It will consist of three blocks of code for three different types of algorithms in ALGORITHM column. Each statement that is created will have the form

if **where-condition** then **variable** = **algorithm-dependent-derivation**

The “if” part of our if-then statements will always be the same – it will be the condition stated in the WHERE column of value-level metadata. Based on this condition we will assign the value to the variable from the VARIABLE column, but what that value would be will be determined separately in each block, so let’s look at them in more detail.

SIMPLE ASSIGNMENT

First example (shown in Picture 4) illustrates the simplest case – the variable is assigned a value from another column CTVALUE. This case is somewhat similar to what we did with %formats macro; however, with value-level metadata you can do more than just assign one variable’s value based on another one. You can now base your variable on several other variables as you can have as many conditions in the WHERE column as you want.

The first block of code in our DATA _NULL_ step will be as follows:

```
if algorithm = "Set to value in CTVALUE" then do;
  call symput ("vlm", symget("vlm")||' if '||strip(where)||" then "
                                    ||strip(variable)||"=");
  if datatype="text" then call symput ("vlm", symget("vlm")||'put("'||strip(ctvalue)
                                                ||'",'||strip(put(length, 3.))||".); ");
  else call symput ("vlm", symget("vlm")||strip(ctvalue)||"; ");
end;
```

The code above shows how we can transform the spreadsheet from Picture 4 into SAS statements. At the each iteration of the data step we add another line of code into our macro variable VLM. Depending on the DATATYPE, we create either a character or a numeric variable.

Below is the value of VLM macro variable after this step:

```
if LBCAT='CHEMISTRY' and LBTESTCD='ALB' then PARAM=put("Albumin (g/dL)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='ALP' then PARAM=put("Alkaline Phosphatase
(U/L)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='ALT' then PARAM=put("Alanine Aminotransferase, ALT
(U/L)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='AST' then PARAM=put("Aspartate Aminotransferase,
AST (U/L)",40.);
if LBCAT='HEMATOLOGY' and LBTESTCD='BASO' then PARAM=put("Basophils (10^9/L)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='BILI' then PARAM=put("Total Bilirubin
(mg/dL)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='CA' then PARAM=put("Total Calcium (mg/dL)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='CHOL' then PARAM=put("Total Cholesterol
(mg/dL)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='CL' then PARAM=put("Chloride (mmol/L)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='CREAT' then PARAM=put("Creatinine (mg/dL)",40.);
if LBCAT='HEMATOLOGY' and LBTESTCD='EOS' then PARAM=put("Eosinophils (10^9/L)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='GLUC' then PARAM=put("Glucose (mg/dL)",40.);
if LBCAT='HEMATOLOGY' and LBTESTCD='HCT' then PARAM=put("Hematocrit (%)",40.);
if LBCAT='HEMATOLOGY' and LBTESTCD='HGB' then PARAM=put("Hemoglobin (g/dL)",40.);
if LBCAT='CHEMISTRY' and LBTESTCD='K' then PARAM=put("Potassium (mmol/L)",40.);
if LBCAT='HEMATOLOGY' and LBTESTCD='LYM' then PARAM=put("Lymphocytes (10^9/L)",40.);
```

ROUNDED VARIABLE VALUE

Another example of algorithm that can be specified in value-level metadata is setting a variable to the value of another variable. If this was all of the derivation, we could have used variable-level metadata for this case, but sometimes we want to have specific number of decimals for each where condition – this is when we can use VLM. Below is a screenshot of value-level metadata for ADLB variable AVAL that is derived as LBSTRESN with a specific number of decimals (which is stored in NUMB_DEC_PLACES column):

DATASET	VARIABLE	WHERE	ALGORITHM	DATA TYPE	LENGTH	NUMB_DEC_PLACES
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='ALB'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='ALP'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='ALT'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='AST'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='HEMATOLOGY' and LBTESTCD='BASO'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	2
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='BILI'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='CA'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='CHOL'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='CL'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='CREAT'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='HEMATOLOGY' and LBTESTCD='EOS'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	2
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='GLUC'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	0
ADLB	AVAL	LBCAT='HEMATOLOGY' and LBTESTCD='HCT'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='HEMATOLOGY' and LBTESTCD='HGB'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='CHEMISTRY' and LBTESTCD='K'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	1
ADLB	AVAL	LBCAT='HEMATOLOGY' and LBTESTCD='LYM'	Set to value of LB.LBSTRESN with specified number of decimals	float	8	2

Picture 5. Value-level metadata example for ADLB.AVAL

With the code below we can create a series of corresponding SAS statements that will round the value of LBSTRESN and place this value in AVAL. This will be the second block of code in the DATA _NULL_ step above:

```
else if prxmatch("/(Set to value of \S+ with specified number of decimals)/", algorithm)
then do;
  round = 0.1 ** numb_dec_places;/*this will be the second argument of ROUND function*/
  call symput ("vlm", symget("vlm")||' if '||strip(where)||" then "
                                  ||strip(variable)||"=");
  if datatype="text" then call symput ("vlm", symget("vlm")||'strip(put(round('
    ||strip(scan(substr(algorithm, 17), 2, ". "))||','||strip(put(round, best.))||'), '
    ||strip(put(length, 3.))||'.'||strip(put(numb_dec_places, 1.))||')); ');
  else call symput ("vlm", symget("vlm")||'round('||strip(scan(substr(algorithm, 17),
                               2, ". "))||','||strip(put(round, best.))||'); ');
end;
```

First we select the records where ALGORITHM column contains text in form “Set to value of *some-variable-name* with specified number of decimals” using PRXMATCH function. After that, depending on the type of the variable that we want to create we produce the corresponding statements. If we are creating a character variable, then first we need to round the value of the original variable and then use PUT function with the numeric format that has a specified number of decimal points. If we are creating a numeric variable, then we just round the original one with the desired precision.

The statements added to VLM macro variable will be as follows:

```
if LBCAT='CHEMISTRY' and LBTESTCD='ALB' then AVAL=round(LBSTRESN,0.1);
if LBCAT='CHEMISTRY' and LBTESTCD='ALP' then AVAL=round(LBSTRESN,1);
if LBCAT='CHEMISTRY' and LBTESTCD='ALT' then AVAL=round(LBSTRESN,1);
if LBCAT='CHEMISTRY' and LBTESTCD='AST' then AVAL=round(LBSTRESN,1);
if LBCAT='HEMATOLOGY' and LBTESTCD='BASO' then AVAL=round(LBSTRESN,0.01);
if LBCAT='CHEMISTRY' and LBTESTCD='BILI' then AVAL=round(LBSTRESN,0.1);
if LBCAT='CHEMISTRY' and LBTESTCD='CA' then AVAL=round(LBSTRESN,0.1);
if LBCAT='CHEMISTRY' and LBTESTCD='CHOL' then AVAL=round(LBSTRESN,1);
if LBCAT='CHEMISTRY' and LBTESTCD='CL' then AVAL=round(LBSTRESN,1);
if LBCAT='CHEMISTRY' and LBTESTCD='CREAT' then AVAL=round(LBSTRESN,0.1);
if LBCAT='HEMATOLOGY' and LBTESTCD='EOS' then AVAL=round(LBSTRESN,0.01);
if LBCAT='CHEMISTRY' and LBTESTCD='GLUC' then AVAL=round(LBSTRESN,1);
if LBCAT='HEMATOLOGY' and LBTESTCD='HCT' then AVAL=round(LBSTRESN,0.1);
if LBCAT='HEMATOLOGY' and LBTESTCD='HGB' then AVAL=round(LBSTRESN,0.1);
if LBCAT='CHEMISTRY' and LBTESTCD='K' then AVAL=round(LBSTRESN,0.1);
if LBCAT='HEMATOLOGY' and LBTESTCD='LYM' then AVAL=round(LBSTRESN,0.01);
```

MULTIPLICATION

As it was mentioned before, value-level metadata is more commonly used in ADaM, however it can be used in SDTM as well. Let’s use these SDTM.LB value-level derivations as another example of where automation can be done. The value of LBSTRESN is defined as LBORRES multiplied by some conversion factor depending on test name and sometimes units:

DATASET	VARIABLE	WHERE	ALGORITHM	DATATYPE	LENGTH
LB	LBSTRESN	TESTNAME='GLUCOSE' and UNITS = 'MG/DL'	Set to value of LBORRES*0.0555	num	8
LB	LBSTRESN	TESTNAME='HDL-CHOLESTEROL'	Set to value of LBORRES*0.0259	num	8
LB	LBSTRESN	TESTNAME='HEMATOCRIT'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='HEMOGLOBIN'	Set to value of LBORRES*10	num	8
LB	LBSTRESN	TESTNAME='IRON'	Set to value of LBORRES*0.1791	num	8
LB	LBSTRESN	TESTNAME='LDL-CHOLESTEROL'	Set to value of LBORRES*0.0259	num	8
LB	LBSTRESN	TESTNAME='LYMPHOCYTES'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='MCHC'	Set to value of LBORRES*10	num	8
LB	LBSTRESN	TESTNAME='MONOCYTES'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='NEUTROPHILS'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='PHOS.'	Set to value of LBORRES*0.323	num	8
LB	LBSTRESN	TESTNAME='RDW'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='REACTIVE LYMPHOCYTES'	Set to value of LBORRES*0.01	num	8
LB	LBSTRESN	TESTNAME='TOT. PROT.'	Set to value of LBORRES*10	num	8
LB	LBSTRESN	TESTNAME='TOTAL BILIRUBIN'	Set to value of LBORRES*17.1037	num	8
LB	LBSTRESN	TESTNAME='TOTAL IRON BINDING CAPACITY'	Set to value of LBORRES*0.1791	num	8
LB	LBSTRESN	TESTNAME='TRIG.'	Set to value of LBORRES*0.0113	num	8
LB	LBSTRESN	TESTNAME='UNSATURATED IRON BINDING CAPACITY'	Set to value of LBORRES*0.1791	num	8
LB	LBSTRESN	TESTNAME='URIC ACID'	Set to value of LBORRES*59.4849	num	8
LB	LBSTRESN	TESTNAME='VLDL(CALCULATED)'	Set to value of LBORRES*0.0259	num	8

Picture 6. Value-level metadata example for LB.LBSTRESN

We can transform such derivations into code as well, which will be the third block in our DATA _NULL_ step:

```
else if prxmatch("/(Set to value of \S+ \* \d+)/", algorithm) then do;
  if prxmatch("/(\w+\.\w+ \*)/", algorithm)
  then var_orig = scan(substr(algorithm, 17), 2, ". ");
  else var_orig = scan(substr(algorithm, 17), 1, " ");
```

```
  call symput ("vlm", symget("vlm")||' if '||strip(where)||' then '||strip(variable)||
               '=input('||strip(var_orig)||',best.) * '||scan(algorithm, -1, " ")||'; ');
end;
```

Using PRXMATCH again, we select the following pattern in ALGORITHM column: “Set to value of *some-variable-name * conversion-factor*”. Now, in the previous example we assumed that variable name is in a two-level format: *dataset-name.variable-name*. Here, we add another check to see whether we have a reference to dataset name or not. With another PRXMATCH we determine whether there was a “.” before “*” – hence, whether there was a two-level variable name in the specification. After we have the actual variable name, we create the SAS statement where our new variable is assigned original variable’s value multiplied by different conversion factor for each WHERE condition.

The following statements will be written to VLM macro variable:

```
if TESTNAME='GLUCOSE' and UNITS = 'MG/DL' then LBSTRESN=input(LBORRES,best.) * 0.0555;
if TESTNAME='HDL-CHOLESTEROL' then LBSTRESN=input(LBORRES,best.) * 0.0259;
if TESTNAME='HEMATOCRIT' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='HEMOGLOBIN' then LBSTRESN=input(LBORRES,best.) * 10;
if TESTNAME='IRON' then LBSTRESN=input(LBORRES,best.) * 0.1791;
if TESTNAME='LDL-CHOLESTEROL' then LBSTRESN=input(LBORRES,best.) * 0.0259;
if TESTNAME='LYMPHOCYTES' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='MCHC' then LBSTRESN=input(LBORRES,best.) * 10;
if TESTNAME='MONOCYTES' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='NEUTROPHILS' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='PHOS.' then LBSTRESN=input(LBORRES,best.) * 0.323;
if TESTNAME='RDW' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='REACTIVE LYMPHOCYTES' then LBSTRESN=input(LBORRES,best.) * 0.01;
if TESTNAME='TOT. PROT.' then LBSTRESN=input(LBORRES,best.) * 10;
if TESTNAME='TOTAL BILIRUBIN' then LBSTRESN=input(LBORRES,best.) * 17.1037;
if TESTNAME='TOTAL IRON BINDING CAPACITY' then LBSTRESN=input(LBORRES,best.) * 0.1791;
if TESTNAME='TRIG.' then LBSTRESN=input(LBORRES,best.) * 0.0113;
if TESTNAME='UNSATURATED IRON BINDING CAPACITY' then LBSTRESN=input(LBORRES,best.) *
0.1791;
if TESTNAME='URIC ACID' then LBSTRESN=input(LBORRES,best.) * 59.4849;
if TESTNAME='VLDL(CALCULATED)' then LBSTRESN=input(LBORRES,best.) * 0.0259;
```

The macro variable is created, so now we just need to use it. All you have to do is call %vlm macro in your dataset program and then invoke VLM macro variable within the appropriate data step:

```
%vlm(domain = LB,
     spec   = "your-location\SDTM_Specs.xlsx");

data lb;
  set rawdata.labs;
  …
  some-statements
  …
  &vlm.;
  …
  some-more-statements
  …
run;
```

It will resolve to all of the statements that were created from value-level metadata and you will have your code generated for you in a single macro call.

Important thing to note is that these if-then statements will be executed in the order in which they were created, which will be the same as the order that they were stated in “Value-Level Metadata” spreadsheet in the specification. So when you are creating that spreadsheet make sure that if, for example, you have a derivation for PARAMCD in value-level metadata, then all derivations that reference PARAMCD in WHERE column go after it.

CONCLUSION

As you can see, possibilities for automation are everywhere. We have discussed the most common areas where macros can be used to implement industry standards and at the same time to make programmers’ lives a little easier. The particular macros that were introduced could be used the way they are or customized for your specific needs. What is more, with this detailed description of how these macros were created and what logic was used during their development you will be able to create similar macros of your own that can widen the scope of automation as far as your fantasy goes.

REFERENCES

1. CDISC Submission Data Standards Team, 2018, “Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.3.”(https://www.cdisc.org/standards/foundational/sdtmig/sdtmig-v3-3)
2. CDISC Analysis Data Model Team, 2016, “Analysis Data Model Implementation Guide Version 1.1.” (https://www.cdisc.org/standards/foundational/adam)
3. CDISC Controlled Terminology (https://www.cdisc.org/standards/terminology)
4. Wendi L. Wright , 2007, Creating a Format from Raw Data or a SAS® Dataset (https://support.sas.com/resources/papers/proceedings/proceedings/forum2007/068-2007.pdf)

ACKNOWLEDGMENTS

I would like to thanks my colleagues Oleksandr Babych and Anna Panasenko for their valuable input and advice, and also for bearing with me while I was writing this.

CONTACT INFORMATION

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

Anastasiia Khmelnytska
Intego Group LLC
19 Hromadyanska Street
Kharkiv 61057, Ukraine
Work Phone: +1 407.512.1006 (ext. 2443) | +380 44.500.7020 (ext. 2443)
Email: anastasiia.khmelnytska@intego-group.com
Web: www.intego-group.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.