# Metadata-based Auto-Programming Process

Lisa Lyons, Janssen R&D US, Qian Zhao, Harry Chen, Janssen R&D China


## ABSTRACT

Traditionally the process for programming ADaM datasets is cumbersome and heavily relies on manual work. We spend most of the time to develop or update code according to analysis specifications. More often than not, study algorithms from previous studies are contained in various programs and locations. Further, it can be a challenge to manage an algorithm library if algorithms are in text format.

This paper shares an approach to automatically generate SAS® code to create ADaM data from source SDTM data via metadata.

The key concept includes extracting algorithms from code to be populated into metadata and managing the metadata to generate the code.

This technique is used in ADaM data creation and can be extended to other uses.


## INTRODUCTION

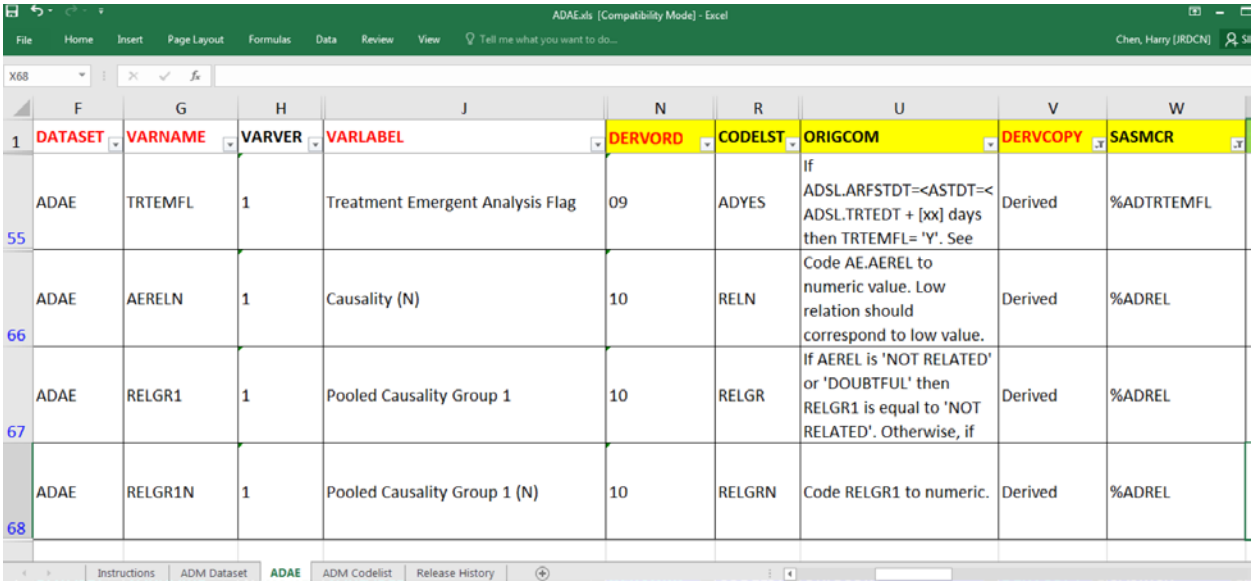
With regulatory agencies enforcing more standardized data and industry moving towards standardizing analysis requirements, we look to achieve more automation for analysis by reusing standard and previous study algorithms stored in metadata to generate SAS code.

Using a modular approach in our ADaM dataset design we will illustrate the concept of metadata based auto programming using ADaM planning specifications, SDTM data and macro and parameter level metadata. We will also provide an example Data Step Level for IF-THEN/ELSE logic, using a metadata approach.

**EXAMPLE 1:** Macro Level Metadata

In this first example, we will demonstrate how to automatically generate the ADAE.sas by using our ADaM Excel planning sheet (specifications) as input, extracting information from standard macro and program folders and converting it to macro level metadata. This macro level metadata is then used to automatically build modularized ADaM SAS code.

**ADAE.xls Specification:** The ADAE.xls specification includes key information such as SAS macro name for variable derivations (SASMCR) and variable derivation order (DERVORD) necessary to create modularized SAS code according to the input specification.



ADAE.xls [Compatibility Mode] - Excel

| | F | G | H | J | N | R | U | V | W |
|---|---|---|---|---|---|---|---|---|---|
| 1 | DATASET | VARNAME | VARVER | VARLABEL | DERVORD | CODELST | ORIGCOM | DERVCOPY | SASMCR |
| 55 | ADAE | TRTEMFL | 1 | Treatment Emergent Analysis Flag | 09 | ADYES | If ADSL.ARFSTDT=<ASTDT=< ADSL.TRTEDT + [xx] days then TRTEMFL= 'Y'. See | Derived | %ADTRTEMFL |
| 66 | ADAE | AERELN | 1 | Causality (N) | 10 | RELN | Code AE.AEREL to numeric value. Low relation should correspond to low value. | Derived | %ADREL |
| 67 | ADAE | RELGR1 | 1 | Pooled Causality Group 1 | 10 | RELGR | If AEREL is 'NOT RELATED' or 'DOUBTFUL' then RELGR1 is equal to 'NOT RELATED'. Otherwise, if | Derived | %ADREL |
| 68 | ADAE | RELGR1N | 1 | Pooled Causality Group 1 (N) | 10 | RELGRN | Code RELGR1 to numeric. | Derived | %ADREL |

Instructions | ADM Dataset | ADAE | ADM Codelist | Release History

***%uacADaMCode* Macro Call Program:** One macro call program *%uacADaMCode* is run by the programmer, which will automatically generate the ADAE.sas code in this example.

```
Remarks             :  The ADMacrodir, studyADPrograms, and domain parameters can
                    :  have multiple separated by a pipe (|).
                    :  For example, &path1 | &path2 for ADMacrodir and studyADPrograms.
                    :  For example, ADAE | ADVS for domain.
Macro Parameters    :  ADMacrodir - paths to reference macro folders
                    :  studyADPrograms -  paths to reference program folders
                    :  metadir - path to your metadata that will be used to create you code
                    :  domain - domains you want to create
                    :  outDir - output directory path write out files
*******************************************************************************/
%uacADaMCode(ADMacrodir = &topdrv.\1_GLOBAL\AnalysisStandards_Test Release\Macros
            ,metadir    = &opath
            ,studyADPrograms = &topdrv.\1_GLOBAL\AnalysisStandards_Test Release\Programs
            ,domain     = ADAE
            ,outDir     = &opath);
```

There are main two steps in the *%uacADaMCode* macro. The first step is to extract information from the macros themselves and save that information in the metadata. The second step, we use the metadata to write out the code.

The *%uacADaMCode* macro will scan the standard macro and program folders to parse the SAS code (by Perl Regular Expression). From the macros, we extract the headers, parameters and default parameter values and convert into macro level metadata. From the standard or study specific program folders, we parse the macro call statements to create macro and parameter level metadata, included in the examples below.

**Macro Level Metadata:**

| macName | paramn | param | equal | paramval |
|---|---|---|---|---|
| %MACRO ADREL | 1 | indsn | = | |
| %MACRO ADREL | 2 | outdsn | = | |
| %MACRO ADREL | 3 | keepvars | = | |
| %MACRO ADREL | 4 | debug | = | OFF |
| %MACRO ADTRTEMFL | 1 | indsn | = | |
| %MACRO ADTRTEMFL | 2 | invar | = | |
| %MACRO ADTRTEMFL | 3 | adrule | = | |
| %MACRO ADTRTEMFL | 4 | outdsn | = | |
| %MACRO ADTRTEMFL | 5 | debug | = | OFF |

**Parameter Level Metadata:** Standard or study specific reference programs are used to help populate the parameter level metadata for each domain.

| domain | macName | paramn | param | equal | paramval |
|---|---|---|---|---|---|
| adae | %ADREL | 1 | indsn | = | adae |
| adae | %ADREL | 2 | outdsn | = | adae |
| adae | %ADREL | 3 | keepvars | = | AERELN RELGR1 RELGR1N |
| adae | %ADREL | 4 | debug | = | OFF |
| adae | %ADTRTEMFL | 1 | indsn | = | adae |
| adae | %ADTRTEMFL | 2 | invar | = | &dayvar |
| adae | %ADTRTEMFL | 3 | adrule | = | VERSION1 |
| adae | %ADTRTEMFL | 4 | outdsn | = | adae |
| adae | %ADTRTEMFL | 5 | debug | = | OFF |

The second step is to generate the macro call statements in the ADaM code. The macro searches the parameter level metadata first. If the specified macro cannot be found in the parameter level metadata, then the default values from macro level metadata is used. The input data name (indsn), output data name (outdsn) and version number (adrule) can also be automatically updated according columns in the planning specification sheet (VARVER corresponds to the standard analysis rule version).

**Autogenerated ADAE.sas**: Below illustrates the section of code that was built for the derived TRTEMFL, AERELN, RELGR1 and RELGR1N variables after applying the parameter and macro level metadata.

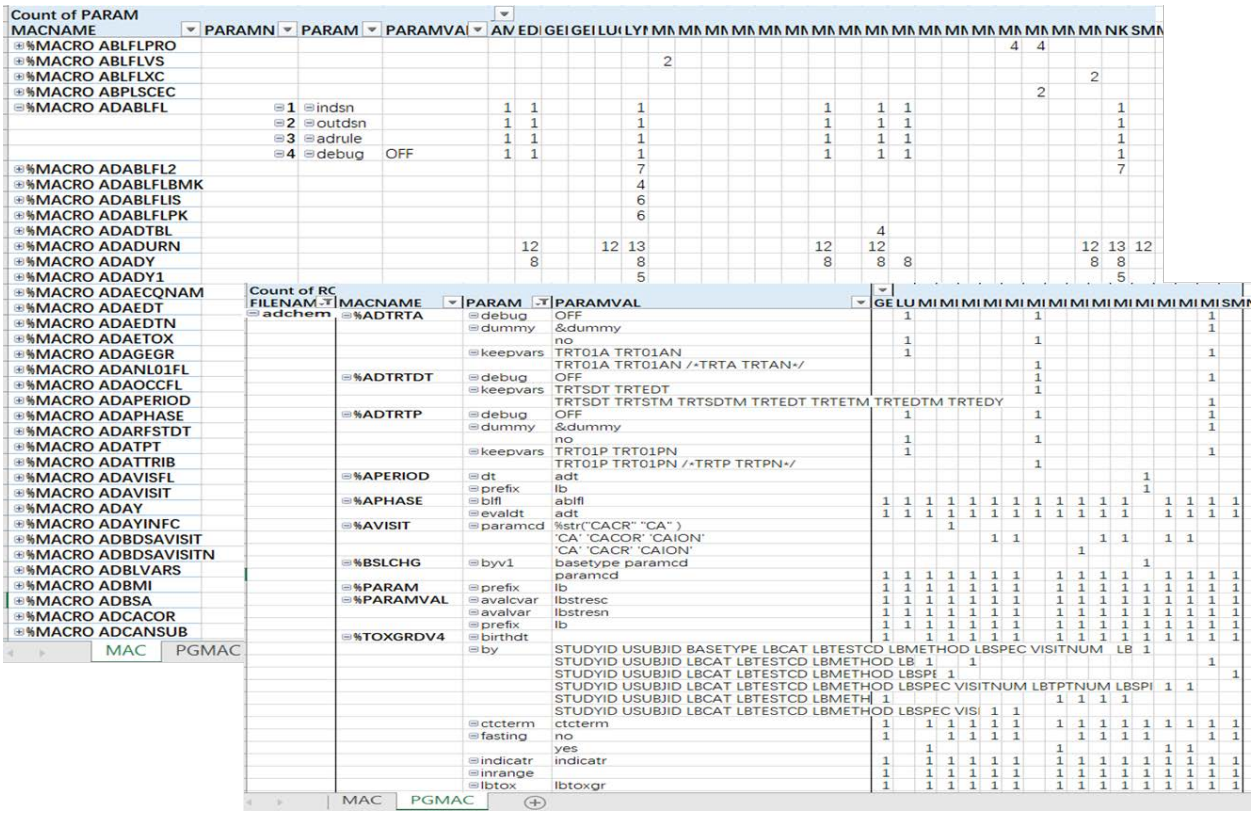
```
/*****************************************************************************
* Short Description: Macro to create Treatment Emergent Analysis Flag
* Input : input dataset indsn,d_in.ae
* Output : input dataset in INDSN with TRTEMFL variable added to create the output dataset in OUTDSN
*****************************************************************************/
%ADTRTEMFL(indsn = _09ADADURN,
           invar=&dayvar,
           adrule = VERSION1,
           outdsn = _10ADTRTEMFL,
           debug=OFF);

/*****************************************************************************
* Short Description: Macro to create Causality & Pooled Causality Group variables
* Input: input dataset is indsn,d_in.ae
* Output: Input dataset in INDSN with AERELN, RELGR1, RELGR1N added to output dataset in OUTDSN  Dependi
*****************************************************************************/
%ADREL(indsn = _10ADTRTEMFL,
       outdsn = _11ADREL,
       keepvars=AERELN RELGR1 RELGR1N,
       debug=OFF);
```

Normally when a new study starts, we manually refer to the programs and macros from a similar study. We might even include different modules from a few studies. Now, we can do this automatically using the *%uacADaMCode* macro call program. It can combine the metadata from multiple previous studies and our standard macro and program library. A key aspect is we benefit and learn from previous experience to select the best macros and algorithms for our current needs.

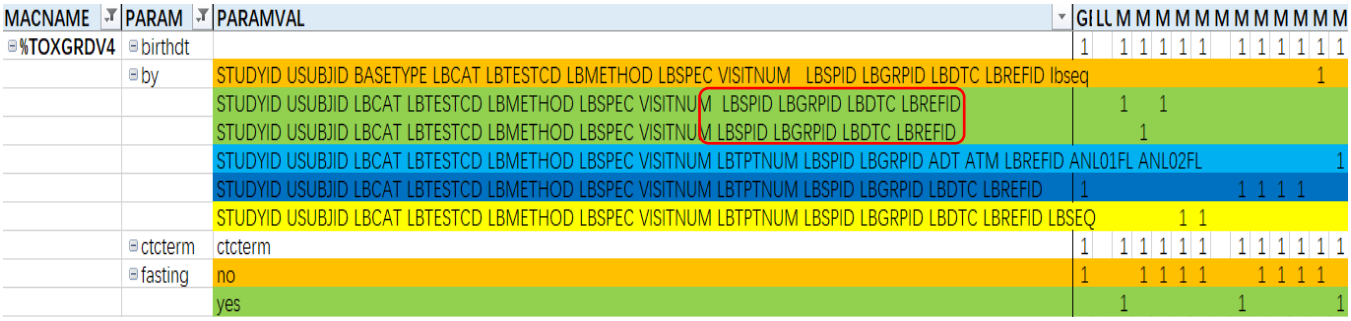
Below illustrates that *%uacADaMCode* macro can reference three previous studies and the standard library to create both the macro and program metadata. We can go straight to the SAS code from here, or as you will see below, we can assess the findings and choose the best options.

```
%uacADaMCode(ADMacrodir = &study1MacroPath | &study2MacroPath| &study32MacroPath| &standardMacroPath
            ,metadir   = &opath
            ,studyADPrograms = &study1ProgramPath | &study2ProgramPath| &study3ProgramPath| &standardProgramPath
            ,domain    = ADSL| ADAE |ADVS |ADLB
            ,outDir    = &opath);
```

Here is an example of a separate macro *%PowerMacroExplorer* we developed that allows you to compare macro and program metadata from multiple studies to obtain a complete overview. In the example below illustrates twenty plus studies under one compound macro level metadata. As we can see, it is easy to find the most commonly used macros and the study-specific macros. Users can learn from the metadata and pick the best one for their own needs and apply it in the *%uacADaMCode* macro.




We know that a similar study design can share similar macro calls. It follows that using the standards and previous studies can save a lot time. Macros have already been tested and validated and users don't need to 're-invent the wheel'. Copying the parameter values from metadata as much as possible can keep the code more consistent. The example below shows that for the 'by' parameter of macro *%TOXGRDV4* there are five different parameter values saved in the comparison metadata. The two lines of parameter values in green are the same but extra spaces caused the inconsistency. Apparently at least one macro parameter value was written manually with an extra space. This allows programmers to see across multiple studies and correct any slight inconsistencies. This allows for macro governance across a disease area or therapeutic areas, which will help ensure robust macro development and consistency.



**EXAMPLE 2:** IF-THEN/ELSE logic in Data Step Level

In this second example, we will demonstrate two cases of how to automatically generate SAS code for complex IF-THEN/ELSE logic in Data Steps.

There is general macro *%ifStatement* developed to create IF-THEN/ELSE statement from metadata in the following ifVar_outVar structure:

Table 1

| groupVar | ifVar1 | ifVar2 | | ••• | ifVarN | _ | outVar1 | outVar2 | ••• | outVarN |
|---|---|---|---|---|---|---|---|---|---|---|
| xxx | xxx | xxx | | xxx | xxx | | xxx | xxx | xxx | xxx |
| xxx | xxx | xxx | | xxx | xxx | | xxx | xxx | xxx | xxx |
| xxx | xxx | xxx | | xxx | xxx | | xxx | xxx | xxx | xxx |

The '_' is the separator column. To the left of the separator are the input variables and to the right are the output variables (the groupVar column is optional). The cells in the table can be filled in with any value, variable name, or even an expression (denoted with a leading hash symbol #), and the IF-THEN/ELSE statement can be generated with a clear nesting structure to map the input variable to the output variable. e.g., the metadata below will generate the code below:

Table 2

| groupVar | ifVar1 | ifVar2 | _ | outVar1 | outVar2 |
|---|---|---|---|---|---|
| 1 | varA | 'valueA' | | 'valueX' | 'valueY' |
| 1 | varA | 'valueB' | | varX | varY |
| 2 | #var1>var2 | 'valueC' | | #expressionX | #expressionY |

```
if  ifVar1=varA then do;
    if   ifVar2='valueA'   then do; outVar1='valueX'; outVar2='valueY';end;
   else if   ifVar2='valueB'   then do; outVar1=varX; outVar2=varY;end;
end;

if   var1>var2    and  ifVar2='valueC'     then do; expressionX; expressionY;end;
```

This approach can be used to convert different specification data into codes.

Assume we have the following specification table in Excel for the lab toxicity grade definition:

| | PARAMCAT | PARAM | LBTESTCD | UNIT | AGELOD | AGEHID | g1lo | g1hi | g2lo | g2hi | g3lo | g3hi | g4lo | g4hi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 1 | 7 | 130<= | <=140 | 120<= | <130 | -9999< | <120 | NA | NA |
| 3 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 8 | 21 | 120<= | <=130 | 100<= | <120 | -9999< | <100 | NA | NA |
| 4 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 22 | 35 | 95<= | <=105 | 80<= | <95 | -9999< | <80 | NA | NA |
| 5 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 36 | 60 | 85<= | <=94 | 70<= | <84 | -9999< | <70 | NA | NA |
| 6 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 61 | 90 | 90<= | <=99 | 70<= | <90 | -9999< | <70 | NA | NA |
| 7 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 91 | 730 | 90<= | <=99 | 70<= | <90 | -9999< | <70 | NA | NA |
| 8 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | 731 | 5840 | 100<= | <=109 | 70<= | <100 | -9999< | <70 | NA | NA |
| 9 | HEMATOLOGY | Neutrophils (x10E9/L) | NEUT | x10E9/L | 1 | 1 | 5<= | <=7 | 3<= | <5 | 1.5<= | <3 | -9999< | <1.5 |
| 10 | HEMATOLOGY | Neutrophils (x10E9/L) | NEUT | x10E9/L | 2 | 6 | 1.75<= | <=2.5 | 1.25<= | <1.75 | 0.75<= | <1.25 | -9999< | <0.75 |
| 11 | HEMATOLOGY | Neutrophils (x10E9/L) | NEUT | x10E9/L | 7 | 60 | 1.2<= | <=1.8 | 0.9<= | <1.2 | 0.5<= | <0.9 | -9999< | <0.5 |
| 12 | HEMATOLOGY | Neutrophils (x10E9/L) | NEUT | x10E9/L | 61 | 90 | 0.75<= | <=1.2 | 0.4<= | <0.75 | 0.25<= | <0.4 | -9999< | <0.25 |
| 13 | HEMATOLOGY | Neutrophils (x10E9/L) | NEUT | x10E9/L | 91 | 5840 | 0.75<= | <=1.2 | 0.4<= | <0.75 | 0.25<= | <0.4 | -9999< | <0.25 |
| 14 | HEMATOLOGY | Platelets (x10E9/L) | PLAT | x10E9/L | 1 | 5840 | NA | NA | 50<= | <=75 | <=25 | <50 | -9999< | <25 |
| 15 | CHEMISTRY | Creatinine (umol/L) | CREAT | umol/L | 1 | 6 | 1*88.4<= | <=1.7*88.4 | 1.7*88.4< | <=2.4*88.4 | 2.4*88.4< | <=3*88.4 | 3*88.4< | <9999 |

The following SAS code can be automatically created:

```
if  PARAMCAT='HEMATOLOGY' then do;
    if  PARAM='Hemoglobin (g/L)' then do;
        if  LBTESTCD='HGB' then do;
            if  UNIT='g/L' then do;
                if  1<=age<=7 then do;
                    if   130<=aval<=140    then  toxgrade=1;
                    else if   120<=aval<130     then  toxgrade=2;
                    else if   -9999<aval<120    then  toxgrade=3;
                end;
                else if  8<=age<=21 then do;
                    /*...*/
                end;
                /*...*/
                else if  731<=age<=5840 then do;
                    if   100<=aval<=109    then  toxgrade=1;
                    else if   70<=aval<100  then  toxgrade=2;
                    else if   -9999<aval<70     then  toxgrade=3;
                end;
            end;
        end;
    end;
    else if  PARAM='Neutrophils (x10E9/L)' then do;
        if  LBTESTCD='NEUT' then do;
            if  UNIT='x10E9/L' then do;
                /*...*/
            end;
        end;
    end;
    else if  PARAM='Platelets (x10E9/L)' then do;
        /*...*/
    end;
end;
else if  PARAMCAT='CHEMISTRY' then do;
    /*...*/
    end;
```

Another example starts with the specifications for the derivation of Best Overall Response per RECIST criteria. Below is an example of the specification.

| Overall response at 1st time point ($X_1$) | Overall response at 2nd time point ($X_2$) | Overall response at 3rd time point ($X_3$) | Best overall Response |
|---|---|---|---|
| CR | Unknown | Unknown | Final: SD if $X_1 \geq XX$ otherwise, NE<br>Interim: If no additional disease assessments are expected* then SD if $X_1 \geq XX$ otherwise, NE else uCR |
| | CR with $X_2 - X_1 \geq YY$ | Any | CR |
| | CR with $X_2 - X_1 < YY$ | Unknown | Final: SD if $X_2 \geq XX$ otherwise, NE<br>Interim: If no additional disease assessments are expected* then SD if $X_2 \geq XX$ otherwise, NE else uCR |
| | | PR | Flag as a data issue with note: cannot be PR after CR.<br>SD if $X_2 \geq XX$ otherwise, PD |
| | | SD | Flag as a data issue with note: cannot be SD after CR.<br>SD if $X_2 \geq XX$ otherwise, PD |
| | | PD | SD if $X_2 \geq XX$ otherwise, PD |
| | PR | Any | Flag as a data issue with note: cannot be PR after CR.<br>SD if $X_1 \geq XX$, otherwise PD |
| | SD | Any | Flag as a data issue with note: cannot be SD after CR.<br>SD if $X_1 \geq XX$, otherwise PD |
| | PD | Any | SD if $X_1 \geq XX$, otherwise PD |
| PR | CR | CR | if $X_3 - X_2 \geq YY$, CR;<br>else it will be changed to leading (PR, CR) sequence due to data Step 10 |
| | Unknown | Unknown | Final: SD if $X_1 \geq XX$ otherwise, NE<br>Interim: If no additional disease |

The specification table in word document converted to a similar structure in a table format in Excel.

| | RS1 | RS2 | X2_X1 | RS3 | X3_X2 | IFC | BOR1 | BOR2 | NOTE | ERROR |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | CR | Unknown | | Unknown | | #X1>=XX | SD | NE | | |
| 3 | CR | CR | #X2-X1>=YY | | | | CR | | | |
| 4 | CR | CR | #X2-X1<YY | Unknown | | #X2>=XX | SD | NE | | |
| 5 | CR | CR | #X2-X1<YY | PR | | #X2>=XX | SD | PD | Note: cannot be PR after CR. | |
| 6 | CR | CR | #X2-X1<YY | SD | | #X2>=XX | SD | PD | Note: cannot be SD after CR. | |
| 7 | CR | CR | #X2-X1<YY | PD | | #X2>=XX | SD | PD | | |
| 8 | CR | CR | #X2-X1<YY | #other | | | | | | #put 'ERROR: Unexpected error 1. ' USUBJID= RSDTC=; |
| 9 | CR | PR | | | | #X1>=XX | SD | PD | Note: cannot be PR after CR. | |
| 10 | CR | SD | | | | #X1>=XX | SD | PD | Note: cannot be SD after CR. | |
| 11 | CR | PD | | | | #X1>=XX | SD | PD | | |
| 12 | CR | #other | | | | | | | | #put 'ERROR: Unexpected error 2. ' USUBJID= RSDTC=; |
| 13 | PR | Unknown | | Unknown | | #X1>=XX | SD | NE | | |
| 14 | PR | CR | | CR | # X3 - X2 >=YY | | CR | | | |
| 15 | PR | CR | | CR | #other | | | | | #put 'ERROR: Unexpected error 3. ' USUBJID= RSDTC=; |

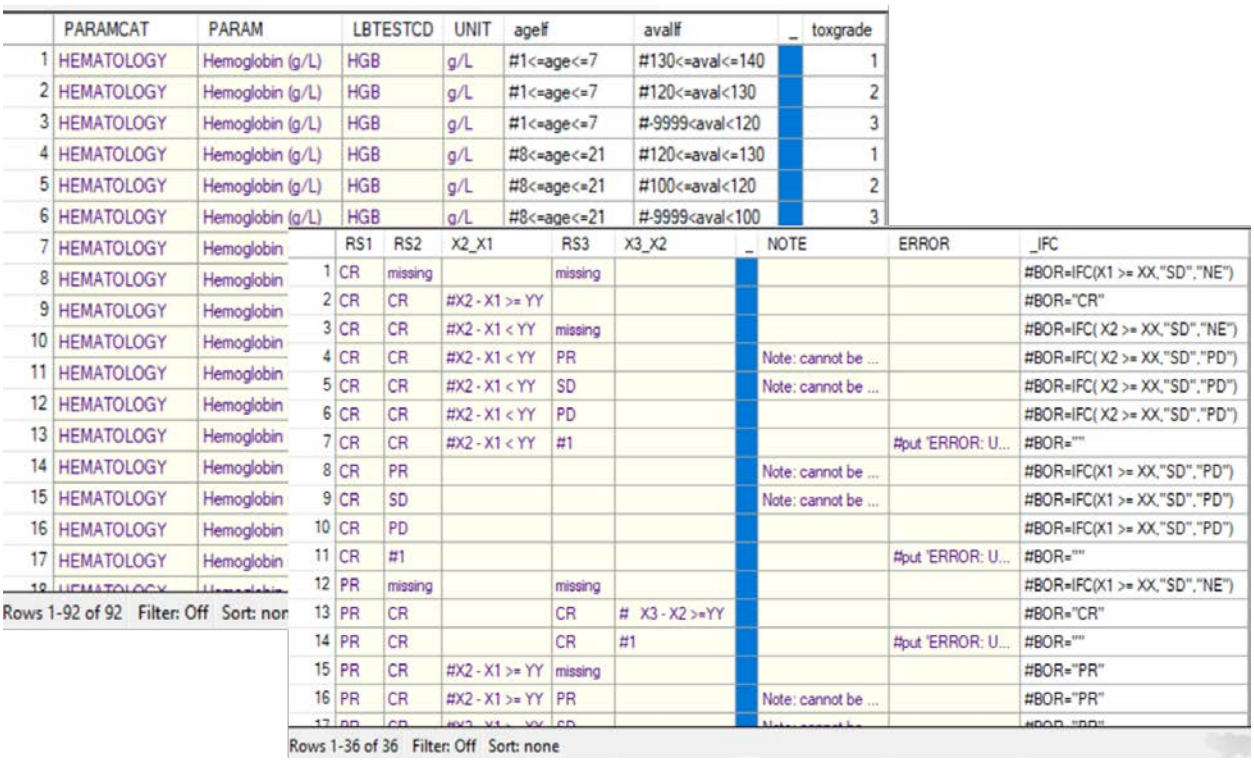
Then the following SAS code can be automatically created:

```
if  RS1='CR' then do;
    if   MISSING(RS2)    and  MISSING(RS3)    then   BOR=IFC(X1 >= XX,"SD","NE");
   else if  RS2='CR' then do;
        if   X2 - X1 >= YY    then   BOR="CR";
       else if  X2 - X1 < YY then do;
            if   MISSING(RS3)    then   BOR=IFC(X2 >= XX,"SD","NE");
           else if   RS3='PR'    then  do; NOTE='Note: cannot be PR after CR.'; BOR=IFC(X2 >= XX,"SD","PD");end;
           else if   RS3='SD'    then  do; NOTE='Note: cannot be SD after CR.'; BOR=IFC(X2 >= XX,"SD","PD");end;
           else if   RS3='PD'    then   BOR=IFC(X2 >= XX,"SD","PD");
           else  do; put 'ERROR: Unexpected error 1. ' USUBJID= RSDTC=;BOR="";end;
       end;
   end;
   else if   RS2='PR'    then  do; NOTE='Note: cannot be PR after CR.'; BOR=IFC(X1 >= XX,"SD","PD");end;
   else if   RS2='SD'    then  do; NOTE='Note: cannot be SD after CR.'; BOR=IFC(X1 >= XX,"SD","PD");end;
   else if   RS2='PD'    then   BOR=IFC(X1 >= XX,"SD","PD");
   else  do; put 'ERROR: Unexpected error 2. ' USUBJID= RSDTC=;BOR="";end;
end;
else if  RS1='PR' then do;
    if   MISSING(RS2)    and  MISSING(RS3)    then   BOR=IFC(X1 >= XX,"SD","NE");
   else if  RS2='CR' then do;
            if  RS3='CR' then do;
                if   X3 - X2 >=YY  then  BOR="CR";
               else  do; put 'ERROR: Unexpected error 3. ' USUBJID= RSDTC=;BOR="";end;
           end;
```

For the previous two use cases, although the input specification is different structure, we only need some simple codes below to convert them to the general ifVar_outVar structure (Table 1):

```
array glo{*} g1lo g2lo g3lo g4lo;
array ghi{*} g1hi g2hi g3hi g4hi;
do i = 1 to dim(glo);
    ageIf = cats('#',agelod,'<=age<=', agehid);
    avalIf = cats('#',glo[i],'aval',ghi[i]);
    _ = '';
    toxgrade = i;
    if (glo[i] ne 'NA' or ghi[i] ne 'NA') then output;
end;
```

```
_BOR1 = quote(strip(BOR1));
_BOR2 = quote(strip(BOR2));
if IFC = '' then _IFC = cats('#BOR=',_BOR1);
else _IFC = cats(tranwrd(IFC,'#','#BOR=IFC('), ',',_BOR1, ',',_BOR2, ')');
```



| | PARAMCAT | PARAM | LBTESTCD | UNIT | ageIf | avalIf | _ | toxgrade |
|---|---|---|---|---|---|---|---|---|
| 1 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #1<=age<=7 | #130<=aval<=140 | | 1 |
| 2 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #1<=age<=7 | #120<=aval<130 | | 2 |
| 3 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #1<=age<=7 | #-9999<aval<120 | | 3 |
| 4 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #8<=age<=21 | #120<=aval<=130 | | 1 |
| 5 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #8<=age<=21 | #100<=aval<120 | | 2 |
| 6 | HEMATOLOGY | Hemoglobin (g/L) | HGB | g/L | #8<=age<=21 | #-9999<aval<100 | | 3 |
| 7 | HEMATOLOGY | Hemoglobin | | | | | | |
| 8 | HEMATOLOGY | Hemoglobin | | | | | | |
| 9 | HEMATOLOGY | Hemoglobin | | | | | | |
| 10 | HEMATOLOGY | Hemoglobin | | | | | | |
| 11 | HEMATOLOGY | Hemoglobin | | | | | | |
| 12 | HEMATOLOGY | Hemoglobin | | | | | | |
| 13 | HEMATOLOGY | Hemoglobin | | | | | | |
| 14 | HEMATOLOGY | Hemoglobin | | | | | | |
| 15 | HEMATOLOGY | Hemoglobin | | | | | | |
| 16 | HEMATOLOGY | Hemoglobin | | | | | | |
| 17 | HEMATOLOGY | Hemoglobin | | | | | | |

Rows 1-92 of 92 Filter: Off Sort: nor

| | RS1 | RS2 | X2_X1 | RS3 | X3_X2 | _ | NOTE | ERROR | _IFC |
|---|---|---|---|---|---|---|---|---|---|
| 1 | CR | missing | | missing | | | | | #BOR=IFC(X1 >= XX,"SD","NE") |
| 2 | CR | CR | #X2 - X1 >= YY | | | | | | #BOR="CR" |
| 3 | CR | CR | #X2 - X1 < YY | missing | | | | | #BOR=IFC( X2 >= XX,"SD","NE") |
| 4 | CR | CR | #X2 - X1 < YY | PR | | | Note: cannot be ... | | #BOR=IFC( X2 >= XX,"SD","PD") |
| 5 | CR | CR | #X2 - X1 < YY | SD | | | Note: cannot be ... | | #BOR=IFC( X2 >= XX,"SD","PD") |
| 6 | CR | CR | #X2 - X1 < YY | PD | | | | | #BOR=IFC( X2 >= XX,"SD","PD") |
| 7 | CR | CR | #X2 - X1 < YY | #1 | | | | #put 'ERROR: U... | #BOR="" |
| 8 | CR | PR | | | | | Note: cannot be ... | | #BOR=IFC(X1 >= XX,"SD","PD") |
| 9 | CR | SD | | | | | Note: cannot be ... | | #BOR=IFC(X1 >= XX,"SD","PD") |
| 10 | CR | PD | | | | | | | #BOR=IFC(X1 >= XX,"SD","PD") |
| 11 | CR | #1 | | | | | | #put 'ERROR: U... | #BOR="" |
| 12 | PR | missing | | missing | | | | | #BOR=IFC(X1 >= XX,"SD","NE") |
| 13 | PR | CR | | CR | # X3 - X2 >=YY | | | | #BOR="CR" |
| 14 | PR | CR | | CR | #1 | | | #put 'ERROR: U... | #BOR="" |
| 15 | PR | CR | #X2 - X1 >= YY | missing | | | | | #BOR="PR" |
| 16 | PR | CR | #X2 - X1 >= YY | PR | | | Note: cannot be ... | | #BOR="PR" |

Rows 1-36 of 36 Filter: Off Sort: none

Once we have the above ifVar_outVar table (Table 1 structure) created, we call the macro:

```
%ifStatement(outSASFile="&_currentPath\ifstatement.sas"
            , indsn = ifMetaData
            , separatorVar = _
            , addQuote = Y
            , nGroupVar = 0);
```

then the IF-THEN/ELSE logic code can be created automatically (here no groupVar needed and the whole table is one group);

The benefits of this approach are:

- IF-THEN/ELSE logic is a powerful, logical construct that can cover many data derivations that may or may not be achieved using other SAS techniques. We can specify value, variable and even expressions in the metadata table.
- Maintain consistency between the specification and the program; therefore, changes need to be made only in one place. The code generated is easy to review and understand.
- Allow making changes in a central location and obtain better traceability from the source variable to the mapped variable.
- Enable a higher-level standardization due to reusability of the mapping specification template and the macro.
- The metadata may be saved in a general structured database to store most of the variable mapping algorithms to build a library

**SUMMARY:**

We can try to achieve as much automation as possible in code generation for analysis, although 100% of automation is not yet possible. Currently the source, output data, specifications and SAS macros are getting more and more standardized. The benefits of Metadata-based Auto-Programming Process are:

- Reduce manual work and potential human error which inevitably leads to wasted time in both coding and debugging.
- Programmer focuses on the logical design instead of detailed technical coding.
- Able to compare with standard/historical algorithms by metadata comparison to enhance the standards and maintain consistency
- Centralized control of coding instead of scattered pieces
- Potentially require lower level of SAS skills.
- Further benefits with future Database and Interface like MDR (MetaData Repository) system

For simplicity of this paper, the two examples are simplified cases using simple SAS macros and Excel data. The concept demonstrated can be extended to more complicated cases with other programming tools, Database and Interface like MDR (Metadataa Repository) system.

**REFERENCES**

1. Qian Zhao, Jun (John) Wang, Ruofei Hao, Generic Macros for Data Mapping, https://www.pharmasug.org/proceedings/2015/BB/PharmaSUG-2015-BB11.pdf
2. Bob Zhong, Jiangfan Li, Hong Xie, Peter De Porre, Kenneth Maahs, Kyounghwa Bae, SAS Macro for Derivation of Best Overall Response per RECIST 1.1, https://www.lexjansen.com/pharmasug/2017/AD/PharmaSUG-2017-AD24.pdf

**CONTACT INFORMATION**

Special thanks to the Janssen Autocode team who contributed to the macros described in this paper *Nick Masel, David Fan, Ilse Augustyns, Phyllis Wolf, Sumesh Kalappurakal, Taku Uryu, James Li, Sendil Ramabadran and Sandy Lei.* Your comments and questions are valued and encouraged. Contact the authors at:

Lisa Lyons
Janssen R&D
1125 Trenton Harbourton Rd.
Titusville, NJ 08640
Email: llyons3@its.jnj.com

Qian Zhao and Harry Chen
Janssen China R&D
65 Gui Qing Lu
Shanghai 200231 China
Email: QZhao1@its.jnj.com,
Email: hchen104@its.jnj.com

Brand and product names are trademarks of their respective companies.