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.
%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.

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:

<table>
<thead>
<tr>
<th>macName</th>
<th>param</th>
<th>param</th>
<th>equal</th>
<th>paramval</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MACRO ADREL</td>
<td>1</td>
<td>indsn</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADREL</td>
<td>2</td>
<td>outdsn</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADREL</td>
<td>3</td>
<td>keepvars</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADREL</td>
<td>4</td>
<td>debug</td>
<td>=</td>
<td>OFF</td>
</tr>
<tr>
<td>%MACRO ADTRLTEMFL</td>
<td>1</td>
<td>indsn</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADTRLTEMFL</td>
<td>2</td>
<td>invar</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADTRLTEMFL</td>
<td>3</td>
<td>adrule</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADTRLTEMFL</td>
<td>4</td>
<td>outdsn</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>%MACRO ADTRLTEMFL</td>
<td>5</td>
<td>debug</td>
<td>=</td>
<td>OFF</td>
</tr>
</tbody>
</table>

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

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.
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.

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

<table>
<thead>
<tr>
<th>groupVar</th>
<th>ifVar1</th>
<th>ifVar2</th>
<th>...</th>
<th>ifVarN</th>
<th>_</th>
<th>outVar1</th>
<th>outVar2</th>
<th>...</th>
<th>outVarN</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
</tr>
<tr>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
</tr>
<tr>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td>xxx</td>
<td></td>
<td>xxx</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>groupVar</th>
<th>ifVar1</th>
<th>ifVar2</th>
<th>_</th>
<th>outVar1</th>
<th>outVar2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>varA</td>
<td>'valueA'</td>
<td>'valueX'</td>
<td>'valueY'</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>varA</td>
<td>'valueB'</td>
<td>varX</td>
<td>varY</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>#var1&gt;var2</td>
<td>'valueC'</td>
<td>#expressionX</td>
<td>#expressionY</td>
<td></td>
</tr>
</tbody>
</table>

```sas
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:
The following SAS code can be automatically created:

```sas
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<=avVal<=140 then toxgrade=1;
          else if 120<=avVal<130 then toxgrade=2;
          else if -9999<avVal<120 then toxgrade=3;
        end;
        else if 8<=age<=21 then do;
          /*...*/
        end;
      end;
    end;
  end;
else if PARAM='Neutrophils (x10^9/L)' then do;
  if LBTESTCD='NEUT' then do;
    if UNIT='x10E9/L' then do;
      /*...*/
    end;
  end;
else if PARAM='Platelets (x10^9/L)' then do;
  /*...*/
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.
The specification table in word document converted to a similar structure in a table format in Excel.

Then the following SAS code can be automatically created:
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):

```plaintext
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:
if _B0R1_ = quote(strip(_B0R1_));
_B0R2_ = quote(strip(_B0R2_));
if _IFC_ = '"' then _IFC_ = cats('#_B0R1_ ', _B0R1_);
else _IFC_ = cats(tranwrd(_IFC_,'#', '#_B0R1_ ', _B0R1_,' ',_B0R1_,' ',_B0R2_,' '));
```
Once we have the above ifVar_outVar table (Table 1 structure) created, we call the macro:

```sas
%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 (Metadata Repository) system.

REFERENCES


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, hchen104@its.jnj.com

Brand and product names are trademarks of their respective companies.