PharmaSUG 2018 - Paper BB-19

Speed Up SDTM/ADaM Dataset Development with If-Less Programming

Lei Zhang and John Lu, Celgene Corporation, Berkeley Height, NJ

ABSTRACT

One of major tasks in building FDA submission packages for a clinical study is to create SDTM/ADaM datasets from CDISC compliant specifications. Most of programmers will find this task tedious, time-consuming, and prone to error because the variables in the specifications often have to be derived from different datasets with complex if-else statements, and updated, and tested with ever evolving datasets. In order to alleviate this dreary process, this paper introduces the if-less programming technique that helps programmers to create SDTM/ADaM variables with fewer or no if-else statements. By applying this technique, a programmer can not only write pieces of derivation code shorter and quicker, but also make them easier to understand, modify, and reuse. What's more, this technique will greatly reduces the programming time to create ASDTM/ADaM datasets and at the same improves the quality of SDTM/ADaM datasets generated.

BACKGROUND

The STDM (Study Data Tabulation Model) and ADaM (Analysis Data Model) is one of the most important CDISC components in clinical trial submission to FDA and other healthcare authorities. They describe the principles and ways to create study tabulation and analysis datasets and associated metadata for submission. Moreover, in the *Study Data Technical Conformance Guide* published in March 2016[1], FDA recommends that the software programs used to create all ADaM datasets should be provided along with the tables and figures in order to help reviewers to better understand how the datasets, tables and figures were created and interacted. The guide aims to help reviewers to further understand the process by which the variables for the respective analyses were created and to confirm the analysis algorithms adopted. Although FDA doesn't require the programs submitted to be run directly under its given environment, the programs for ADaM datasets are treated as part of important documents for submission, and they have to be well-written, easy to read and understand at least.

However, writing good programs to build SDTm/ADaM datasets is not an easy task. Like developing ADaM specification, most of time programmers find it a dreary process. This is because each SDTM/ADaM dataset in a clinical study often has ten, or more derived variables that have to be coded with numerous if-else (including select-when) statements. Excessive use of if-else statements usually makes a program hard to read, test and update. Besides, since the SDTM/ADaM specification is a living document, an ADaM program full of control statements is often difficult to extend, modify, and reuse with the evolving specification, or in other studies.

In order to understand this issue well, let's take a look at a simple ADaM specification example. Below are the abridged definitions of 7 ADSL variables from a one treatment open-label mockup study.

ID	Variable Name	Variable Label	Source/Derivation	Notes
1	TRT01A	Actual Treatment for Period 01	ENRLFL=Y then 'Drug A' else Blank.	ENRLFL is enrollment flag.
2	TRT01AN	Actual Treatment for Period 01 (N)	ENRLFL=Y then 1 = 'Drug A' else 2 = BLANK	
3	SAFFL	Safety Population Flag	if ENRLFL=Y and TRTSDA is not missing, then set to Y; otherwise, set to N.	TRTSDA is treatment start date, or first dose date
4	DSFL	Discontinuation Flag	if DSDT is not missing then set to Y Otherwise set to N	
5	INVNAM	Investigator Name	Derive from SITEID - 002 = 'PARIS' 010 = 'NICE' 011 = 'NEW YORK" 013 = 'WASHINGTON' 021 = 'BEIJING'	SITEID is the site ID
6	COUNTRY	Country	Convert SITEID to numeric - if 1<= and < =10 then set to 'FRA'; if 10< and <=20 then set to 'USA'; if >20 then set to 'CHN'; otherwise set to BLANK.	
7	LSTCTDT	Last contact date	First non-missing date among the death date(DTHDT), discontinuation date(DISCDT), last dose date(LSTDDT) + 14 days, and last visit date(LSTVDT).	

As you may notice, the 7 ADaM derived variables described above are piecewise functions of one, or more variables in the underlying datasets. They can be implemented within the following typical data step if-else or select-when statements:

```
DATA ADSL;
       ...

       /* TRT01A */
      if ENRLFL='Y' then TRT01A='Drug A';
       else TRT01A=' ';

       /* TRT01AN */
      if ENRLFL='Y' then TRT01AN=1;
       else TRT01AN=2;

       /* SAFFL */
       if ENRLFL='Y' and not missing(TRTSDA) then SAFFL='Y';
       else SAFFL='N';

       /* DSFL */
       If not missing(DSDT) then DSFL='Y';
       else DSFL='N';

       /* INVNAM */
       select (siteid);
          when ("002") INVNAM='PARIS';
          when ("010") INVNAM='NICE';
          when ("011") INVNAM='NEW YORK';
          when ("013") INVNAM='WASHINGTON';
          when ("021") INVNAM='BEIJING';
          otherwise INVNAM=' ';
       end;

       /* COUNTRY */
       if 1 <= input(SITEID,8.) <=10 then COUNTRY='FRA';
       else if 10 < input(SITEID,8.) <=20 then COUNTRY='USA';
       else if input(SITEID,8.) >20 then COUNTRY='CHN';
        else COUNTRY=' ';

       /* LSTCTDT */
       if nmiss(DTHDT)=0 then LSTCTDT=DTHDT;
       else if nmiss(DISCDT)=0 then LSTCTDT=DISCDT;
       else if nmiss(LSTDDT)=0 then LSTCTDT= LSTDDT+14;
       else LSTCTDT=LSTVDT;
       ...
```

As you can see, numerous if-else/select-when statements are used in order to create those derived variables, and the program ends up with many branches with verbose code. This causes following concerns:

- It makes the program difficult to reason about because of code branches introduced by if-else statements.
- It makes the program difficult to test or debug. You have to test each of branches in the code.
- It makes the program hard to update or extend because there are chances that the difference in behavior between branches can be quite big.
- It causes some unintended side effects, because code in an if-else block has to change a variable defined outside the branch, thus the code in different branches may change the variable inconsistently.

Next section, I'll introduce the if-less programming techniques that enable you to derive SDTM/ADaM variables using short and concise expressions instead of the cumbersome if-else/select-when statements.

WHAT IS IF-LESS PROGRAMMING

Like many other procedural programming language, the if-else statement is fundamental to SAS, and we certainly don't want to get rid of it in all the programs, but often you have to acknowledge that an if-else statement causes more time and effort to write and understand than an assignment statements [2][3][4]. There are at least three benefits if the if-else statements can be minimized or avoided in a program:

- The program without if-else statements is easier to read
- The program without if-else statements is easier to test
- The program without if-else statements is easier to update or extend.

So the question become how we can use less if-else statements in the SAS programs. Fortunately, SAS has already provided a few of alternative ways to help programmers to deal with that.

One of approaches is to replace a list of if-else statements with built-in function PUT() and INPUT() with the help of informats or formats. For example, variable INVNAM and COUNTRY in the above example can be derived with following alternative code:

```
PROC FORMAT;
invalue $sitefmt
"002"='PARIS'
"010"='NICE'
"011"='NEW YORK'
"013"='WASHINGTON'
"021"='BEIJING'
Other=' '
;
Value cntyfmt
1 - 10 ='FRA'
10< - 20 ='USA'
20< - High='CHN'
OTHER=' '
;
Run;

DATA ADSL;
       ...

       /* INVNAM */
       INVNAM=input(SITEID, $SITEFMT.);

       /* COUNTRY */
       COUNTRY=PUT(input(SITEID,8.),CNTYFMT.);

       ...
RUN;
```

This is quite a popular method to replace if-else statements, especially when you have a large number of values that have to be mapped or transformed simply, but this method has following limitations:

- It only works with limited cases. For example, variable SAFFL can't be derived with this method.
- The format or informat referenced has to be defined elsewhere first, either in a separate code segment, or even in another program file, which causes code fragments, and adds extra layer to the code understanding.
- It may be overkilled sometimes. For example, if variable TRT01A, and TRT01AN were coded with this method, the programmer would end up with writing longer pieces of code than usual.

The second method is to use the function IFN() and IFC() to replace if-else statements, and CHOOSEN() and CHOOSEC() to replace select-when statements. This set of built-in functions have been introduced since SAS 9, for more information about them, please see [5]. For example, TRT01A and INVNAM can be implemented with following code:

```
DATA ADSL;
  ...
       /* TRT01AN */
  TRT01AN=IFN(ENRLFL='Y', 1, 2);

       /* TRT01A */
  TRT01A=IFC(TRT01AN=1, 'Drug A',' ');

       /* INVNAM */
  INVNAM=CHOOSEC(1+(SITEID="002")*1 + (SITEID="010")*2
         + (SITEID="011")*3 + (SITEID="013")*4
     + (SITEID="021")*5,
     ' ', 'PARIS', 'NICE', 'NEW YORK', 'WASHINGTON', 'BEIJING'
        );

  ...
RUN;
```

Although using those 4 functions have some advantages over the if-else or select-when statements[6], there are some problems with it as well:

- IFN() and IFC() work well as a replacement of simple if-else statements, but not with nested if-else statements. For example, if using IFC(), the COUNTRY variable will have to be coded with nested IFC() function calls as follows, which makes code much less readable and modifiable.

```
DATA ADSL;
       ...;
       /* COUNTRY */
  COUNTRY=IFC(1<= input(SITEID,8.) <=10, 'FRA'
   ,IFC(10 < input(SITEID,8.) <=20, 'USA'
   ,IFC(input(SITEID,8.) >20,'CHN',' ')));

       ...;
RUN;
```

- CHOOSEC() and CHOOSEN() sometimes can't replace the select-when statements straightforward. They often have to be coded with a little trick like the one in the alternative implementation of variable INVNAM. The code is not easy to understand without a second thought.

Can we have a better way to replace if-else or select-when statements in both simple and complex SDTM/ADaM variable derivations? In the next section, I'll describe the third method, which is accomplished by the mapping macros that are essentially the generalization of the function IFN(), IFC(), CHOOSEN(), and CHOOSEC().

DEVELOPING MAPPING MACROS TO SUPPORT IF-LESS PROGRAMMING

Since the function IFN(), IFC(), CHOOSEN(), and CHOOSEC() can perform if-else, or select-when logic and map a value into another in a concise way in simple situations, it is very meaningful to make it generic so that nested or complex logics can be coded in a simple and succinct way as well. This can be accomplished through following 6 mapping macros I am going to introduce below.

Inline Macro Function	Purpose
%N2WRD()	Maps a numeric value to a character value.
%N2N()	Maps a numeric value to another numeric value.
%COND2N()	Maps a condition or a logical expression to a numeric value.
%WRD2N()	Maps a character value to a numeric value.
%WRD2WRD()	Maps a character value to another character value.
%COND2WRD()	Maps a condition or a logical expression to a character value.

This set of macros are developed as the inline macro functions for data steps, which are the composites of IFN(), IFC(), CHOOSEN(), and CHOOSEC(). They can be used like any built-in data step function. Each of the mapping macros enables you to apply a list of inline mapping rules against all values in one, or more variables and returns a new variable that contains all mapped values, thus offering programmers an elegant way to create a variable based on the result of multiple comparisons. As you will see later, the 6 mapping macros provide a clear, expressive, and efficient way to encode derived variables without using any if-else statements.

%N2WRD

%N2WRD() maps a numeric value to a character one. As an extension of function CHOOSEC(), it accepts varying number of arguments with following syntax.

Syntax

```
%N2WRD(
 nvalue /* Numeric expression to be searched in the mapping list*/
 ,<n-identifier-1#>c-value-1 /* Mapping item 1 */
 ,<n-identifier-1#>c-value-2 /* Mapping item 2 */
 , , ,
 ,<n-identifier-k#>c-value-k /* Mapping item k */
 ,OTHER=' ' /* Character value returned if no match */
 ,SEP=#   /* Separator between identifier and value part of a mapping item */
)
```

The first positional parameter of the mapping macro (that is, `nvalue`) is required as a numeric value to be searched in the inline mapping list which is consisted of the rest of positional arguments. Each mapping item in the mapping list contains two parts: the first part is the numeric identifier of the item, and the second part is the character value of the item to be returned when its identifier equals `nvalue`. The character # is used as a separator between those two parts by default. You can change it to a different separator with key parameter SEP=. If the identifier of a mapping item is not provided, then the sequence number (or the item number)of the mapping item will be used. %N2WRD() checks the inline mapping list from left to right until it finds the first item with its identifier equals `nvalue`. In case none of identifiers is equal to `nvalue`, it returns the value provided by key parameter OTHER=.

The following example demonstrates the use of %N2WRD():

```
DATA N2WRD;
 Length phase1 phase2 $20;
 input n@@;
 phase1=%N2WRD(n,"Yes","No", Other="Missing");
 phase2=%N2WRD(n,2:"Yes",-1:"No", Other="Missing", sep=:);
 put "Output: " n= phase1= phase2=;
datalines;
1 2 -1 5.1 .
;
RUN;
```

The results in the log:

```
Output: n=1 phase1=Yes phase2=Missing
Output: n=2 phase1=No phase2=Yes
Output: n=-1 phase1=Missing phase2=No
Output: n=5.1 phase1=Missing phase2=Missing
Output: n= phase1=Missing phase2=Missing
```

%N2N

%N2N() maps a numeric value to another numeric value. As an extension of function CHOOSECN(), it is similar to %N2WRD() in both syntax and usage except that it returns a numeric value instead of character one. Below is its syntax:

Syntax

```
%N2N(
 nvalue /* Numeric expression to be searched in the mapping list*/
 ,<n-identifier-1#>n-value-1 /* Mapping item 1 */
```

```
 ,<n-identifier-1#>n-value-2 /* Mapping item 2 */
 , , ,
 ,<n-identifier-k#>n-value-k /* Mapping item k */
 ,OTHER=.  /* Numeric value returned if no match */
 ,SEP=#   /* Separator between identifier and value part of a mapping item */
)
```

The following example demonstrates its use:

```
DATA N2N;
       input n@@;
       m1=%N2N(n,20,35,50,100,other=-99);
       m2=%N2N(n,5:20,4:35,3:50,2:100,other=-88, sep=:);
       put "Output: " n= m1= m2=;
datalines;
1 2 3 4 5 .
;
RUN;
```

The results in the log:

```
Output: n=1 m1=20 m2=-88
Output: n=2 m1=35 m2=100
Output: n=3 m1=50 m2=50
Output: n=4 m1=100 m2=35
Output: n=5 m1=-99 m2=20
Output: n= m1=-99 m2=-88
```

%COND2N

%COND2N() maps a condition or logical expression to a numeric value. All of its arguments are used to form an inline logical mapping list. Each item in the logical mapping list also contains two part parts (separated by # by default). The first part is the identifier that is a logical expression, or condition, and the second part is the numeric value to be returned when the associated identifier is evaluated to TRUE. Like %N2WRD() or %N2N(), it has two key parameters OTHER= and SEP=. %COND2N() has following syntax:

Syntax

```
%COND2N(
 logical-identifier-1<#n-value-1> /* Mapping item 1 */
 ,logical-identifier-2<#n-value-2> /* Mapping item 2 */
 , , ,
 ,logical-identifier-k<#n-value-k> /* Mapping item k */
 ,OTHER=. /* Value-returned when all identifiers are evaluated to FALSE */
 ,SEP=#   /* Separator between identifier and value part of a mapping item */
)
```

%COND2N() checks the inline logical mapping list, from left to right, until it finds the first mapping item with its logical identifier being evaluated to TRUE. If none of identifiers are evaluated to TRUE, then the numeric value in key parameter OTHER= is returned. If a logical mapping item is provided without the value part, %COND2N() takes the sequence number of the mapping item as the value to be returned. Below is an example that demonstrates its uses.

```
DATA COND2N;
 input x@@;
 n=%COND2N(0<=x<5, 5<=x<10#110, (x=10)#120, x>=10, other=-1);
 put "Output: " x= n=;
datalines;
1.5 5 10 15.5 -10 .
;
RUN;
```

The results in the log:

```
Output: x=1.5 n=1
```

```
Output: x=5 n=110
Output: x=10 n=120
Output: x=15.5 n=4
Output: x=-10 n=-1
Output: x= n=-1
```

%WRD2N

%WRD2N() is the reverse of %N2WRD(), but a little more complicated. It maps a character value to a numeric one with following syntax:

Syntax

```
%WRD2N (
 cvalue   /* Character value to be searched in the mapping list */
 ,c-identifier-1<#n-value-1> /* Mapping Item 1 */
 ,c-identifier-2<#n-value-2> /* Mapping item 2 */
 , , ,
 ,c-identifier-k<#n-value-k> /* Mapping item k */
 ,OTHER=.  /* Numeric value-returned if no match */
 ,SEP=#   /* Separator between identifier and value part of a mapping item */
 ,MOD=' '  /* Comparison modifiers similar to those defined in compare()*/
 )
```

The first positional parameter (that is, `cvalue`) is required as a character value to be searched in the inline mapping list that is consisted of the rest of positional arguments in the macro. Each item in the mapping list contains two parts (separated by # by default) : the first part is the character identifier of the item, and the second part is the numeric value of the item to be returned when its associated identifier matches with `cvalue`. Like %N2WRD(), if the value part of a mapping item is not provided, then the sequence number of the mapping item is used as the value to be returned. %WRD2N() checks the inline mapping list from left to right and returns the value of the first identified mapping item.

The following example demonstrates the use of %WRD2N():

```
DATA WRD2N;
 Length s $16;
 input s $ 1-16;
 n1=%WRD2N(s,"Yes","No", Other=-9, MOD=":");
 n2=%WRD2N(s,"Y":2,"N":1, Other=-8, MOD=":i", SEP=:);
 put "Output: " s= n1= n2=;
datalines;
Yes
No
YES
NO
Y
N

;
RUN;
```

The results in the log:

```
Output: s=Yes n1=1 n2=2
Output: s=No n1=2 n2=1
Output: s=YES n1=-9 n2=2
Output: s=NO n1=-9 n2=1
Output: s=Y n1=1 n2=2
Output: s=N n1=2 n2=1
Output: s= n1=-9 n2=-8
```

Please note in case none of identifiers matches with `cvalue`, it returns the numeric value provided in key parameter OTHER=. What is more, %WRD2N() allows a programmer to compare the identifier of a mapping item with `cvalue` using the modifiers provided by key parameter `MOD=`, which are similar to those defined in the function COMPARE(). For example, in the above example, MOD=':' is used to instruct %WRD2N() to truncate the longer of two character

values to be compared to the length of the shorter one. You can also use other modifiers in comparison if needed, such as 'I', 'N', ,'L', or their combinations.

%WRD2WRD

%WRD2WRD() maps a character value to another character one. It has following syntax:

Syntax

```
%WRD2WRD(
 cvalue  /* Character value to be searched in the mapping list*/
 ,c-identifier-1<#c-value-1> /* Mapping Item 1 */
 ,c-identifier-2<#c-value-2> /* Mapping Item 2 */
 , , ,
 ,c-identifier-k<#c-value-k> /* Mapping Item k */
 ,OTHER=' ' /* Character value to be returned when no match */
 ,SEP=#   /* Separator between identifier and value part of a mapping item */
 ,MOD=' '  /* Comparison modifiers similar to those defined in compare()   */
)
```

Like %WRD2N(), %WRD2WRD() takes a varying number of arguments. The first positional argument (that is `cvalue`) of the mapping macro provides the character value to be searched in the inline mapping list that is consisted of the rest of positional arguments. Each item in the mapping list has two parts: the first part is a character identifier to be compared with `cvalue`, and the second part is the character value to be returned when its identifier matches with `cvalue`. The character # is used by default as a separator between the two parts. It can be replaced with other character using key parameter `SEP=`. Similar to %WRD2N(), %WRD2WRD() allows you to control how to compare identifiers with `cvalue` using key parameter `MOD=`. In case the value part of a mapping item is not provided, then the sequence number (in character) of the mapping item is used as the value part. Below is a code example that demonstrates its uses.

```
DATA WRD2WRD;
 Length code1 code2 $20 s $16;
 input s $ 1-16;
 code1=%WRD2WRD(s,"Yes","No", Other="UNK", mod="i");
 code2=%WRD2WRD(s,"Yes"#"Y","No"#"N", Other="UNK", mod=":");
 put "Output: " s= code1= code2=;
datalines;
Yes
No
Yes, Sir
Y
N

;
RUN;
```

The results in the log:

```
Output: s=Yes code1=1 code2=Y
Output: s=No code1=2 code2=N
Output: s=Yes, Sir code1=UNK code2=Y
Output: s=Y code1=UNK code2=Y
Output: s=N code1=UNK code2=N
Output: s= code1=UNK code2=UNK
```

%COND2WRD

%COND2WRD() maps a conditional or logical expression to a character value. It takes a varying number of arguments that form an inline logical mapping list. Each item in the logical mapping list has two part parts (separated by # by default). The first part is the identifier consisting of a logical expression, and the second part is the character value to be returned when the associated identifier is evaluated to TRUE. It has following syntax:

Syntax

```
%COND2WRD(
 logical-identifier-1<#c-value-1> /* Mapping Item 1 */
 ,logical-identifier-2<#c-value-2 /* Mapping Item 2 */
 , , ,
 ,logical-identifier-k<#c-value-k> /* Mapping Item k */
 ,OTHER=' ' /* Character value to be returned when all are evaluated to FALSE */
 ,SEP=#   /* Separator between the identifier-k and selection-k       */
)
```

%COND2WRD() checks the inline mapping list, from left to right, until it finds the first mapping item whose associated identifier is evaluated to TRUE, and then return its value part. If no logical identifiers are evaluated to TRUE, it then returns the value provided in the key parameter OTHER=. In addition, %COND2WRD() allows you to provide a logical expression only in a mapping item. In that case, the sequence number (in character)of the mapping item is returned if the identifier is evaluated to TRUE. Like other mapping macros, %COND2WRD() use # as default separator between the logical expression and value part. When needed, it allows you to choose a different character as a separator with SEP=. Below is a code example that demonstrates its uses.

```
DATA COND2WRD;
       Length code $20;
       input x@@;
 code=%COND2WRD(0<=x< 5#"LT 5", x<10#"5 GE, LT 10"
     ,x>=10#"GT 10", other="Missing");
 put "Output: " x= code=;
datalines;
1.5 5 10 15.5 -10 .
;
RUN;
```

The results in the log:

```
Output: x=1.5 code=LT 5
Output: x=5 code=5 GE, LT 10
Output: x=10 code=GT 10
Output: x=15.5 code=GT 10
Output: x=-10 code=5 GE, LT 10
Output: x= code=5 GE, LT 10
```

Rewriting the Example with Mapping Macros

With the above 6 mapping macros, let's see how the ADSL variables previously defined can be re-implemented:

```
DATA ADSL;
       ...

       /* TRT01AN */
       TRT01AN=%WRD2N(ENRLFL,'Y',OTHER=2);
       /* TRT01A */
       TRT01A=%N2WRD(TRT01AN,1#'Drug A',OTHER=' ');
       /* SAFFL */
       SAFFL=%COND2WRD((ENRLFL='Y' and not missing(TRTSDA))#'Y'
         ,OTHER='N');
       /* DSFL */
      DSFL=%COND2WRD(NMISS(DISCDT)=0#'Y', OTHER='N');
       /* INVNAM */
       INVNAM=%WRD2WRD(SITEID
         ,"002"#'PARIS',"010"#'NICE'
         ,"011"#'NEW YORK',"013"#'WASHINGTON'
         ,"021"#'BEIJING', OTHER=' ');
       /* COUNTRY */
       COUNTRY=%COND2WRD(1<= input(SITEID,8.) <=10#'FRA'
            ,10 < input(SITEID,8.) <=20#'USA'
            ,input(SITEID,8.) >20#'CHN'
               ,OTHER=' ');
```

```
        /* LSTCTDT */
        LSTCTDT=%COND2N(
        (nmiss(DTHDT)=0)#DTHDT
            ,(nmiss(DISCDT)=0)#DISCDT
            ,(nmiss(LSTDDT)=0)#LSTDDT+14
            ,OTHER=LSTVDT);
        ...
RUN;
```

It is clear that using the mapping macros to derive ADaM variables has following benefits:

- It is easy to infer how the code works.
- The code is clear, concise and expressive, having all the required information together in one place.
- Unnecessary code blocks and branching are eliminated, or reduced to their most refined forms.
- The code is easy to test and validate because the mapping macros are stateless and have no side effects.

OTHER IF-LESS PROGRAMMING TRICKS

The mapping macros introduced above help SDTM/ADaM programmers avoid roughly 70% of if-else and select-when statements when coding ADaM derived variables. However, there are more coding techniques that enable programmers to reduce the uses of if-else or select-when statements in a progarm. One is to use other built-in SAS functions, such as COALESCE, PRX, MIN and MAX functions. For example, the function COALESCEN() can be used to select the first non-missing numeric value among several ones, therefore you can derive the variable LSTCTDT with following simple code:

```
LSTCTDT=COALESCE(DTHDT,DISCDT,LSTDDT+14,LSTDT);
```

Using logical expressions is another technique. For example, if you want to derive the common variable ASTDY defined below,

Derivation ID	Variable Name	Variable Label	Source/Derivation
	ASTDY	Analysis Start Relative Day	ASTDT – ADSL.TRTSDT + 1 if ASTDT is on or after TRTSDT, else ASTDT – ADSL.TRTSDT if ASTDT precedes TRTSDT

you can write a piece of code by combination with a logical expression like this

```
ASTDY = ASTDT – TRTSDT + (ASTDT >= TRTSDT);
```

Another technique worthy to mention is to create user-defined functions with PROC FCMP, and call them from data steps. This technique is especially useful when you have a common set of algorithms to be implemented for more than one ADaM variables. Here is an example. Assuming that we want to create an analysis start date variable ASTDT and the correspond flag ASTDTF for ADAE datasets using following imputation method:

Derivation ID	Variable Name	Variable Label	Source/Derivation	Notes
1	ASTDT	Analysis Start Date	Derive from AE.AESTDTF - impute partial dates using the following rules: Missing Year: No imputation, return missing value Missing day and month: impute the day and month with 'June 15', return the numeric date. Missing day only: impute the day with '15', return the numeric date	In ISO8601
2	ASTDTF	Analysis End Date Imputation Flag	Derived from AE.AESTDTF. If it is completely missing or no missing then ASTDTF=' '; otherwise, if start date has day and month missing then ASTDTF='M' else if start date has day missing then ASTDTF='D'.	In ISO8601

The imputation method involves a few of if-else statements, and it can be implemented with the mapping macros as follows.

```
DATA ADAE;
        . . .
   /* ASTDTF */
```

```
       ASTDTF=%COND2WRD(
     lengthn(AESTDTF)=6#'D'
    ,lengthn(AESTDTF)=4#'M'
        ,OTHER=' '
    );

       /* ASTDT */
       ASTDT=input(
      %WRD2WRD(ASTDTF,
         ,"D"#cats(AESTDTF,"15")
         ,"M"#cats(AESTDTF,"0701")
         ,OTHER=AESTDTF)
      ,B8601DA.) ;
 . . .
Run;
```

However, since the imputation flag variable ASTDTF will have to be created by other ADaM datasets, such as ADLB, ADCM, it is better to implement it as a resuable user-defined function to further simplify ADaM programming across multiple datasets. Below is the sample code:

```
PROC FCMP Outlib=WORK.usrfunc.string;
 /* Get the ADaM date imputation flag */
 Function ImpFlag(
  CDate $ /* Character expression in ISO8601 */
 )$1;
 Return(%COND2WRD(
     lengthn(CDATE)=6#'D'
    ,lengthn(CDATE)=4#'M'
    ,OTHER=' '
   )
 );
 Endsub;
RUN;
options cmplib=(WORK.usrfunc);

DATA ADAE;
  Length ASTDTF $1 AESTDTF $8;
  . . .
  /* ASTDTF */
  ASTDTF=ImpFlag(AESTDTF);
  . . .
RUN;
```

A user defined function can encapsulate multiple if-else and/or select-when statements, and handle one or more derived variables at one time, thus making the code much concise across multiple ADaM dataset programs.

CONCLUSION

This paper describes the mapping macros along with several other if-less programming techniques that enable programmers to replace if-else or select-when statements with short, concise and expressive code pieces in a program. When it comes to building SDTM/ADaM datasets for regulatory submissions, those techniques are proved to be extremely useful in helping programmers to develop better SDTM/ADaM datasets faster according to a SDTM/ADaM specification. It is expected that more and more programmers will adopt the if-less programming techniques in their SDTM/ADaM dataset creation and other routine programming tasks to gain the maximum benefits from them.

REFERENCE

1. U.S. Food and Drug Administration, "Study Data Technical Conformance Guide v3.0." March 2016. Available at http://www.fda.gov/downloads/ForIndustry/DataStandards/StudyDataStandards/UCM384744.pdf
2. Lisnic, Andrei "If-less programming". Available at http://alisnic.github.io/posts/ifless/

3. Jouan, Matthias “if-else trees vs SOLID principles”. Available at http://blog.mjouan.fr/if-else-trees-vs-solid-principles/
4. McCabe, THOMAS J., “A Complexity Measure”, *IEEE Transactions on Software Engineering*, VOL. SE-2, NO.4, DECEMBER 1976
5. SAS Institute, Inc. “SAS® 9.2 Language Reference: Dictionary, Fourth Edition”. Available at http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000245852.htm
6. Eberhardt, Peter, Shao, Lucheng. “Functioning at a Higher Level: Using SAS® Functions to Improve Your Code”, *Proceedings of the PharmaSUG 2014 Conference*
7. Zhang, Lei “Building Better ADaM Datasets Faster With If-less Programming”, *Proceedings of the WUSS 2016 Conference*

ACKNOWLEDGEMENTS

DISCLAIMER: The opinions expressed in this presentation and on the following slides are solely those of the presenter and not necessarily those of Celgene Corporation. Celgene Corporation does not guarantee the accuracy or reliability of the information provided herein.

CONTACT INFORMATION

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

Lei Zhang
Celgene Corporation.
400 Connell Dr.
Berkeley Heights NJ 07922
Phone: (908) 673-9000

APPENDIX 1: SAMPLE MAPPING MACROS FOR ILLUSTRATIVE PURPOSES

```
/* NOTE: the sample mapping macros only accept a mapping list with up to 8 items */

/*$ N2WRD
 Purpose: Mapping numeric value to character value.
*/
%Macro N2WRD(
   nValue /* Numeric value to be mapped */
  ,ITEM1, ITEM2, ITEM3, ITEM4  /* Mapping items (1 - 8)  */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=' ' /* Value to be returned if no items are identified */
  ,SEP=#   /* Separator between identifier and value part of a mapping item */
);

%LOCAL IDX N K XITEM CMPFN;
%LOCAL ENDFLAG;

%let N=8;
%LET IDX= 1;
%let endflag=0;
%let xitem=&&ITEM&IDX;
%DO %WHILE ((%length(&xitem) ne 0) and (&endflag=0));
  %LET IDX=%EVAL(&IDX + 1);
  %if &IDX=%eval(&N+1) %then %let endflag=1;
  %else %let xitem=&&ITEM&IDX;
```

```
%End;

%let K=%eval(&idx-1);
%if &K=0 %then %do;
  %PUT ERROR: Empty conditional mapping list.;
  %return;
%end;

%let xitem=&&ITEM&K;
%let n_exp=%sysfunc(scan(&xitem,1,&sep,Q));
%let n_sel=%sysfunc(scan(&xitem,2,&sep,Q));
%if %length(&n_sel)=0 %then %do;
       %let n_sel=&n_exp;
       %let n_exp=&K;
%end;
%let CMPFN=IFC((&nValue)=(&n_exp),&n_sel,&OTHER);

%DO idx=%eval(&K-1) %to 1 %by -1;
  %let xitem=&&ITEM&IDX;
  %let n_exp=%sysfunc(scan(&xitem,1,&sep,Q));
  %let n_sel=%sysfunc(scan(&xitem,2,&sep,Q));
       %if %length(&n_sel)=0 %then %do;
              %let n_sel=&n_exp;
              %let n_exp=&idx;
       %end;
  %let CMPFN=IFC((&nValue)=(&n_exp),&n_sel,&CMPFN);
%END;
&CMPFN
%Mend;


/*$ N2N
 Purpose: Mapping numeric value to numeric value.
*/
%Macro N2N(
   nValue /* Numeric value to be mapped */
  ,ITEM1, ITEM2, ITEM3, ITEM4  /* Mapping items (1 - 8)  */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=. /* Value to be returned if no items are identified */
  ,SEP=#  /* Separator between identifier and value part of a mapping item */
);

%LOCAL IDX N K XITEM CMPFN;
%LOCAL ENDFLAG;

%let N=8;
%LET IDX= 1;
%let endflag=0;
%let xitem=&&ITEM&IDX;
%DO %WHILE ((%length(&xitem) ne 0) and (&endflag=0));
  %LET IDX=%EVAL(&IDX + 1);
  %if &IDX=%eval(&N+1) %then %let endflag=1;
  %else %let xitem=&&ITEM&IDX;
%End;

%let K=%eval(&idx-1);
%if &K=0 %then %do;
  %PUT ERROR: Empty conditional mapping list.;
  %return;
%end;

%let xitem=&&ITEM&K;
%let n_exp=%sysfunc(scan(&xitem,1,&sep,Q));
```

```
%let n_sel=%sysfunc(scan(&xitem,2,&sep,Q));
%if %length(&n_sel)=0 %then %do;
       %let n_sel=&n_exp;
       %let n_exp=&K;
%end;
%let CMPFN=IFN((&nValue)=(&n_exp),&n_sel,&OTHER);

%DO idx=%eval(&K-1) %to 1 %by -1;
  %let xitem=&&ITEM&IDX;
  %let n_exp=%sysfunc(scan(&xitem,1,&sep,Q));
  %let n_sel=%sysfunc(scan(&xitem,2,&sep,Q));
       %if %length(&n_sel)=0 %then %do;
             %let n_sel=&n_exp;
             %let n_exp=&idx;
       %end;
  %let CMPFN=IFN((&nValue)=(&n_exp),&n_sel,&CMPFN);
%END;
&CMPFN
%Mend;

/*$ COND2N
 Purpose: Mapping a logical expression, or condition to a numeric value.
*/

%Macro COND2N(
   ITEM1, ITEM2, ITEM3, ITEM4 /* Mapping items (1 - 8) */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=. /* Value to be returned if no items are identified */
  ,SEP=#  /* Separator between identifier and value part of a mapping item */
);

%LOCAL IDX N K XITEM ENDFLAG;
%LOCAL CMPFN L_EXP N_EXP;

%let N=8;
%LET IDX= 1;
%let endflag=0;
%let xitem=&&ITEM&IDX;
%DO %WHILE ((%length(&xitem) ne 0) and (&endflag=0));
  %LET IDX=%EVAL(&IDX + 1);
  %if &IDX=%eval(&N+1) %then %let endflag=1;
  %else %let xitem=&&ITEM&IDX;
%End;

%let K=%eval(&idx-1);
%if &K=0 %then %do;
  %PUT ERROR: Empty conditional mapping list.;
  %return;
%end;

%let xitem=&&ITEM&K;
%let L_EXP=%sysfunc(scan(&xitem,1,&sep,Q));
%let N_EXP=%sysfunc(scan(&xitem,2,&sep,Q));
%if %length(&N_EXP)=0 %then %let N_EXP=&K;
%LET CMPFN=IFN(&L_EXP,&N_EXP,&OTHER);

%DO idx=%eval(&K-1) %to 1 %by -1;
  %let xitem=&&ITEM&IDX;
  %let L_EXP=%sysfunc(scan(&xitem,1,&sep,Q));
  %let N_EXP=%sysfunc(scan(&xitem,2,&sep,Q));
  %if %length(&N_EXP)=0 %then %let N_EXP=&IDX;
  %LET CMPFN=IFN(&L_EXP,&N_EXP,&CMPFN);
%END;
```

```
&CMPFN
%Mend;

/*$ WRD2N
 Purpose: Mapping a character value to a numeric value.
*/
%Macro WRD2N(
  cValue  /* Character value to be mapped */
  ,ITEM1, ITEM2, ITEM3, ITEM4 /* Mapping items (1 - 8) */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=' ' /* Value to be returned if no items are identified */
  ,SEP=# /* Separator between identifier and value part of a mapping item */
  ,MOD=' ' /* comparison modifiers that is same as those */
       /* used by built-in function COMPARE().    */
);

%LOCAL XITEM IDX LOC LOC1;
%LOCAL EXPLST1 EXPLST2;
%LOCAL PART1 PART2;
%LOCAL C_EXPS N_SELS;

%let C_EXPS=;
%let N_SELS=&OTHER;

%LET IDX=1;
%LET XITEM=&&ITEM&IDX;
%DO %WHILE(%LENGTH(&XITEM));

  %let part1=%sysfunc(scan(&xitem,1,&SEP,Q));
  %let part2=%sysfunc(scan(&xitem,2,&SEP,Q));

  %if &idx = 1 %then %DO;
             %let C_EXPS=%str(compare(trimn(&cValue),trimn(&part1),&MOD)=0);
  %end; %else %do;
             %let
C_EXPS=&C_EXPS%str(,)%str(compare(trimn(&cValue),trimn(&part1),&MOD)=0);
       %end;

  %if %length(&Part2)=0 %then %let Part2=&IDX;
  %let N_SELS=&N_SELS%str(,)&part2;

  %LET IDX=%EVAL(&IDX+1);
  %LET XITEM=&&ITEM&IDX;
%END;

%if %length(&C_EXPS) %then %do;
Choosen(1+%COND2N(%unquote(&C_EXPS),OTHER=0),%unquote(&N_SELS))
%end; %else %do;
&OTHER
%end;
%Mend;

/*$ WRD2WRD
 Purpose: Mapping a character value to a character value.
*/
%Macro WRD2WRD(
  cValue /* Character value to be mapped */
  ,ITEM1, ITEM2, ITEM3, ITEM4 /* Mapping items (1 - 8) */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=' ' /* Value to be returned if no items are identified */
  ,SEP=# /* Separator between identifier and value part of a mapping item */
  ,MOD=' ' /* comparison modifiers that is same as those */
       /* used by built-in function COMPARE().    */
```

```
);

%LOCAL XITEM IDX LOC LOC1;
%LOCAL EXPLST1 EXPLST2;
%LOCAL PART1 PART2;
%LOCAL C_EXPS C_SELS;

%let C_EXPS=;
%let C_SELS=&OTHER;

%LET IDX=1;
%LET XITEM=&&ITEM&IDX;
%DO %WHILE(%LENGTH(&XITEM));

  %let part1=%sysfunc(scan(&xitem,1,&SEP,Q));
  %let part2=%sysfunc(scan(&xitem,2,&SEP,Q));

  %if &idx = 1 %then %DO;
             %let C_EXPS=%str(compare(trimn(&cValue),trimn(&part1),&MOD)=0);
  %end; %else %do;
             %let
C_EXPS=&C_EXPS%str(,)%str(compare(trimn(&cValue),trimn(&part1),&MOD)=0);
       %end;

  %if %length(&Part2)=0 %then %let Part2="&IDX";
  %let C_SELS=&C_SELS%str(,)&part2;

  %LET IDX=%EVAL(&IDX+1);
  %LET XITEM=&&ITEM&IDX;
%END;

%if %length(&C_EXPS) %then %do;
Choosec(1+%COND2N(%unquote(&C_EXPS),OTHER=0),%unquote(&C_SELS))
%end; %else %do;
&OTHER
%end;
%Mend;

/*$ COND2WRD
 Purpose: Mapping a logical expression or condition to a character value.
*/
%Macro COND2WRD(
   ITEM1, ITEM2, ITEM3, ITEM4 /* Mapping items (1 - 8) */
  ,ITEM5, ITEM6, ITEM7, ITEM8
  ,OTHER=' ' /* Value to be returned if no items are identified */
  ,SEP=# /* Separator between identifier and value part of a mapping item */
);

%LOCAL XITEM IDX LOC LOC1;
%LOCAL COND_EXPS C_SELS;
%local explst1 explst2;
%local part1 part2;

%let COND_EXPS=;
%let C_SELS=&OTHER;

%LET IDX=1;
%LET XITEM=&&ITEM&IDX;
%DO %WHILE(%LENGTH(&XITEM));

  %let part1=%sysfunc(scan(&xitem,1,&SEP,Q));
  %let part2=%sysfunc(scan(&xitem,2,&SEP,Q));
```

```
  %if &idx = 1 %then %let COND_EXPS=&part1;
  %else %let COND_EXPS=&COND_EXPS%str(,)&part1;

  %if %length(&C_SELS)=0 %then %let C_SELS="&idx";
  %let C_SELS=&C_SELS%str(,)&part2;

  %LET IDX=%EVAL(&IDX+1);
  %LET XITEM=&&ITEM&IDX;
%END;

%if %length(&COND_EXPS) %then %do;
Choosec(1+%cond2n(%unquote(&COND_EXPS),OTHER=0),%unquote(&C_SELS))
%end; %else %do;
&OTHER
%end;
%Mend;
```