Paper CC-167

Converting Non-Imputed Partial Dates (for SDTM Data Sets) Using PROC FCMP

Noory Y. Kim, CROS NT LLC, Chapel Hill, NC

ABSTRACT

SDTM (Study Data Tabulation Model) data sets are required to store date values with ISO8601 formats, which accommodate both complete dates (e.g. *YYYY-MM-DD*) and partial dates (e.g. *YYYY-MM*). On the other hand, raw data sets may come with non-ISO8601 date formats (e.g. *DDMMMYYYY*). Converting complete date values to an ISO8601 format can be as simple as applying a SAS® date format to a numeric version of the date value. Conversion of partial date values is trickier. How may we convert, say, “UNOCT2016” and “UNUNK2016” into “2016-10” and “2016” respectively? This paper provides examples of how this can be done using PROC FCMP, the SAS Function Compiler procedure. This paper also gives examples of how to avoid the output of nonexistent dates such as “2016-01-99”.

INTRODUCTION

Standards by CDISC (Clinical Data Interchange Standards Consortium) specify that SDTM data sets store date values using ISO8601 formats such as *YYYY-MM-DD* and *YYYY-MM*. On the other hand, raw data sets may store date values in other formats such as *DDMMMYYYY* (the SAS format DATE9) or *MM-DD-YYYY*.

Furthermore, the FDA prohibits the imputation of partial dates for SDTM data sets:

> “SDTM should not include any imputed data. If there is a need for data imputation, this should occur in an analysis dataset, and the relevant supporting documentation to explain the imputation methods must be provided.” (FDA, 2011)

This paper provides an example of a PROC FCMP function that converts partial dates without imputing them.

USING THE ASSIGNMENT OPERATOR WITH A PROC FCMP FUNCTION

For those unfamiliar with PROC FCMP, Carpenter (2013) provides a helpful introduction with numerous examples of functions and subroutines.

PROC FCMP allows the SAS programmer to define custom functions using much of the same syntax one can use within a DATA step. Once compiled and located in memory, a PROC FCMP function can be invoked just like a built-in SAS function. Unlike SAS macro functions, PROC FCMP functions can be used to assign values to a new variable using the assignment (=) operator:

```
new_variable = user_defined_function(existing_variable);
```

These functions could be invoked multiple times within the same DATA step, with just one line of code per invocation.

```
new_variable1 = user_defined_function(existing_variable1);
new_variable2 = user_defined_function(existing_variable2);
```

For SDTM data sets, we could convert the start and end dates of a domain such as CM (concomitant medications) with the following invocations, where `_CMSTDTC` and `_CMENDTC` are the respective source variables for `CMSTDTC` and `CMENDTC`, and `convertdate()` is a PROC FCMP function such as the one defined on page 3.

```
CMSTDTC = convertdate(_CMSTDTC);
CMENDTC = convertdate(_CMENDTC);
```

DISPLAYING PARTIAL DATE VALUES IN SDTM DATASETS

Partial dates are usually missing either the day (e.g. “UNOCT2016”) or both day and month (e.g. “UNUNK2016”). Sometimes partial dates may contain information about the day but not the month (e.g. “17UNK2016”). The CDISC SDTM Implementation Guide (SDTMIG) allows for two ways to store such a date value:

> “Missing components are represented by right truncation or a hyphen (for intermediate components that are missing.” (See SDTMIG Version 3.2, Section 4.1.4.2.)

Right truncation ignores all components of finer granularity than the missing component, whereas hyphenation replaces the missing component (but not the hyphens between components) with a single hyphen. Table 1 contrasts the results of the two options.

Source Date Value	ISO8601 Date Value	Option
17UNK2016	2016	Right truncation
17UNK2016	2016---17	Hyphenation

Table 1: Options allowed by CDISC for representing dates with intermediate components missing.

The examples in this paper implement the right truncation option. That is, the following rules will be observed:

- Omit day components if the month value is unknown.
- Omit day and month components if the year value is unknown.

MAIN EXAMPLE

EXPECTED INPUT

The complexity of a function will depend on the variability of the function input. To keep our example relatively simple, let us suppose that the date values in the raw data sets have the following characteristics (after all aberrant values have been passed on as data queries and resolved):

1. Non-missing date values all have 9 characters following the pattern *DDMMMYYYY* (similar in appearance to the numeric date format DATE9).
2. Date values with missing day have in place of *DD* a character string of length 2. For example, “UKJAN2016”.
3. Date values with missing month have in place of *MMM* a character string of length 3 that does not match an English abbreviation for month (e.g. “JAN”, “FEB”, “MAR”, etc.). For example, “UKUNK2016”.
4. Date values with missing year have in place of *YYYY* a character string of length 4 with at least one non-numeric character. For example, “UKUNKUNKN”.

TARGET OUTPUT

Table 2 shows instances of how our function should work. The last two rows of the table illustrate how to apply the right truncation option for representing partial dates.

Input	Output	Rule Followed
17OCT2016	2016-10-17	
17Oct2016	2016-10-17	Ignore case when converting the month component
UNOCT2016	2016-10	
UNUNK2016	2016	
17UNK2016	2016	Omit day component if month is unknown.
17OCTUNKN	*null*	Omit day and month components if year is unknown.

Table 2: Output to be generated by the function

DEFINING AND COMPILING THE FUNCTION

Below is the SAS code that defines a PROC FCMP function named `convertdate()` to implement the conversion rules above:

```
proc fcmp outlib=work.functions.conversions;                              /*[1]*/

   function convertdate(indate $) $;                                      /*[2]*/

      length outdate $10;                                                 /*[3]*/

      if indate ne ' ' then do;

         yyyy = substr(indate, 6, 4);                                     /*[4]*/
         mmm  = upcase(substr(indate, 3, 3));
         dd   = substr(indate, 1, 2);

         /* if year not missing */
         if notdigit(yyyy) = 0 then do;                                   /*[5]*/

            mm = put(mmm, $month.);                                       /*[6]*/

            /* if month not missing */
            if mm ne ' ' then do;

               /* if day not missing */
               if notdigit(dd) = 0 then do;                               /*[5]*/

                  outdate = yyyy || '-' || strip(mm) || '-' || dd;        /*[7]*/

               end; /* if notdigit(dd) = 0 */
               else outdate = yyyy || '-' || strip(mm);

            end; /* if mm ne ' ' */
            else outdate = yyyy;

         end; /* if notdigit(yyyy) = 0 */
         else outdate = ' ';

      end; /* if indate ne ' ' */
      else outdate = ' ';

      return(outdate);                                                    /*[8]*/

   endsub;                                                                /*[9]*/

run;
```

Notes:

[1] Each function has a four level name: *library.dataset.package.function_name*. In this example we use the temporary `work` *library*, name the *dataset* `functions`, and name the *package* `conversions`. Alternatively, the function could be saved to a permanent library instead of `work`.

[2] The FUNCTION statement begins the function definition. The dollar sign ($) inside the parentheses indicates that the input variable *indate* is a character variable. The dollar sign outside the parentheses indicates that the output variable *outdate* (as determined by the `return()` statement below) is a character variable.

[3] To avoid truncation of character variables handled by the function, use a LENGTH statement (just as you would in a DATA step).

[4] These three statements extract year, month, and day with the assumption that the input values come in the form *DDMMMYYYY*.

[5] An IF statement with the `notdigit()` function equal to zero selects only those character strings composed entirely of numbers. (Feeding the `notdigit()` function with a character string having at least one non-numeric character will result in an output greater than zero.)

[6] This line converts the month abbreviation MMM to a number MM using the following format already defined.

```
proc format;
  value $month
    'JAN' = '01'
    'FEB' = '02'
    'MAR' = '03'
    'APR' = '04'
    'MAY' = '05'
    'JUN' = '06'
    'JUL' = '07'
    'AUG' = '08'
    'SEP' = '09'
    'OCT' = '10'
    'NOV' = '11'
    'DEC' = '12'
    other = ' '
  ;
run;
```

Note that the effectiveness of this format depends on the use of the `upcase()` function in a statement associated with note [4].

[7] This step concatenates a complete date. (Later on in this paper we will replace this line of the code to avoid the output of nonexistent dates.)

[8] The RETURN statement specifies what the function will output.

[9] The ENDSUB statement ends the function definition.

IDENTIFYING THE LOCATION OF COMPILED FUNCTIONS

To have compiled functions accessible in the (current) SAS session, include an OPTIONS statement with the names of function libraries you want to use.

```
options cmplib=(library.dataset);
```

In our example, we identify the location of the compiled function with the following statement:

```
options cmplib=(work.functions);
```

INVOKING THE FUNCTION

A PROC FCMP function can be invoked only after it has been compiled and its location identified. It can be invoked from within a DATA step or from within a PROC step that allows the invocation of PROC FCMP functions (e.g. PROC SQL).

```
data one;
   infile cards;
   input date_date9 $9.;
   cards;
17OCT2016
17Oct2016
UNOCT2016
UNUNK2016
17UNK2016
17OCTUNKN
99JAN2016
31FEB2016
;
```

```
data two;
    set one;
    length date_iso8601 $10;
    date_iso8601 = convertdate(date_date9);
run;
```

ACTUAL OUTPUT

The following shows the output of a PROC PRINT step of the resulting data set:

Obs	date_ date9	date_ iso8601
1	17OCT2016	2016-10-17
2	17Oct2016	2016-10-17
3	UNOCT2016	2016-10
4	UNUNK2016	2016
5	17UNK2016	2016
6	17OCTUNKN	
7	99JAN2016	2016-01-99
8	31FEB2016	2016-02-31

Output 1: Actual output from a PROC PRINT statement of data set `two`.

The actual output shown in Output 1 is consistent with the target output. However, sometimes missing day is indicated with a strictly numerical string, e.g. “99”. The `convertdate()` function had not been defined to detect such missingness codes. As a result, the actual output displays nonexistent date values such as “2016-01-99”.

PREVENTING THE OUTPUT OF NONEXISTENT DATES

How may we prevent the output of such nonexistent dates? Aside from submitting queries to data management, omitting the day (e.g. “2016-01” and “2016-02”) might be an option.

Two approaches of modifying the `convertdate()` function to prevent the output of such nonexistent dates are presented below.

METHOD 1: COMPARE WITH THE LAST EXISTING DATE OF THE SAME MONTH

We can modify the `convertdate()` function to prevent nonexistent dates by replacing the line in gray (with footnote reference `[7]`) with the following block of code:

```
year = input(yyyy, 8.);
month = input(mm, 8.);
day = input(dd, 8.);

month_start_date = mdy(month, 1, year);
month_end_date   = intnx('month', month_start_date, 0, 'end');
month_end_day    = day(month_end_date);

if day > month_end_day or day < 1 then outdate = yyyy || '-' || strip(mm);
else outdate = yyyy || '-' || strip(mm) || '-' || dd;
```

This code compares the input date value with the last existing date of the month (as determined by the `intnx()` function). If the input date value has a day value that exceeds the last existing day of that month or a day value less than 1, then the day value is omitted from the output. This method yields the output shown in Output 2 on the following page.

Obs	date_ date9	date_ iso8601
1	17OCT2016	2016-10-17
2	17Oct2016	2016-10-17
3	UNOCT2016	2016-10
4	UNUNK2016	2016
5	17UNK2016	2016
6	17OCTUNKN	
7	99JAN2016	**2016-01**
8	31FEB2016	**2016-02**

Output 2: Output from a PROC PRINT statement of data set `two` after modification of `convertdate()` to prevent the output of nonexistent dates (by either Method 1 or Method 2). Changes from Output 1 are in boldface text.

METHOD 2: CHECK IF CONVERSION TO A NON-MISSING NUMERIC VALUE IS POSSIBLE

Another way to prevent nonexistent dates is to replace the line in gray (with footnote reference `[7]`) with the following block of code, which checks the validity of a character date value by seeing if that value can be converted to a non-missing numeric date value.

```
outdate = yyyy || '-' || strip(mm) || '-' || dd;

outdate_numeric = input(outdate, anydtdte10.);
if outdate_numeric < .z then outdate = yyyy || '-' || strip(mm);
```

This modification yields the same output as the output by Method 1, shown above in Output 2.

In SAS 9.2, this modification also yields the following log warning:

```
WARNING: Unable to load TKFormat for ANYDTDTE10., proceeding with MVA format
definition.
```

According to the SAS Knowledge Base, this type of log warning may be ignored. (See SAS Knowledge Base, Problem Notes 17881 and 20545: http://support.sas.com/kb/17/881.html; http://support.sas.com/kb/20/545.html.)

If we try to remove the warning replacing `anydtdte10` with `??anydtdte10`, the result will be a non-working function with log error messages. In other words, PROC FCMP does not accommodate allow the use of the format modifier `??` (which prevents log error messages when used in a DATA step).

This log warning does not appear in SAS 9.3 or SAS 9.4.

If `anydtdte10` is replaced with `anydtdte`, the output will vary across different versions of SAS. In SAS 9.2, the output will be the same as Output 2. SAS 9.3 and SAS Studio 9,4, on the other hand, will convert “01JAN2016” into “`2016-01`”.

METHOD 1 VERSUS METHOD 2

The author suggests using Method 1 rather than Method 2, since Method 2 can come with some pitfalls when working across different versions of SAS.

CAVEATS

Much but not all DATA step syntax can be used within a FCMP definition. Examples of syntax that cannot be used in an FCMP function are the format modifier `??` and the IN operator. (See SAS Knowledge Base, Problem Note 51685: http://support.sas.com/kb/51/685.html.)

Other differences between what is allowed within a DATA step versus what is allowed within a FCMP definition are noted in the *Base SAS Procedures Guide*.

SUMMARY

DATE VALUES IN SDTM DATA SETS

SDTM data sets are required to store and display date values with ISO8601 formats (such as *YYYY-MM-DD*). When raw data sets come with date values with non-ISO8601 formats, it is necessary to convert these date values into ISO8601 formats. This conversion should be done without imputation in order to comply with FDA guidelines on SDTM data sets.

PROC FCMP

The SAS user can define PROC FCMP functions to carry out routine tasks. Once compiled and located in memory, a PROC FCMP function can be invoked just like a built-in SAS function. Unlike a macro function, a PROC FCMP function can be used with the assignment operator (=).

The internal syntax of a PROC FCMP function is similar to the internal syntax of a DATA step. For example, it is good programming practice to use LENGTH statements in a PROC FCMP function just as you would in a DATA step to avoid the truncation of character variables.

PROC FCMP functions, however, cannot use everything that a DATA step can. Examples of this are the IN operator and the `??` format modifier.

EXAMPLES OF PROC FCMP FUNCTIONS IN THIS PAPER

This paper presented examples of how PROC FCMP can be used to convert complete and partial dates of the form *DDMMMYYY* to an ISO8601 format while avoiding imputation of partial dates.

The PROC FCMP function in the main example used the `notdigit()` function to detect a missing day or year value coded with a character string having least one nonnumeric character, e.g. “UN” and “UNKN”. This function, however, did not detect missing day values coded with a numeric string, e.g. “99”. Modifications of the function in the main example enabled the detection of these numeric string codes, thereby avoiding the output of nonexistent dates. One such modification used the `intnx()` function to determine the last existing day of a particular month.

The author hopes these examples will serve as a useful reference for SAS programmers developing their own PROC FCMP functions.

REFERENCES

- Adams, John H. 2010. “The new SAS 9.2 FCMP Procedure, what functions are in your future?” *Proceedings of the Pharmaceutical SAS Users Group (PharmaSUG) 2010 Conference*. Cary, NC: SAS Institute Inc. Available at http://www.lexjansen.com/pharmasug/2010/ad/ad02.pdf.
- Carpenter, Arthur L. 2013. “Using PROC FCMP to the Fullest: Getting Started and Doing More.” *Proceedings of the Pharmaceutical SAS Users Group (PharmaSUG) 2013 Conference*. Cary, NC: SAS Institute Inc. Available at http://www.pharmasug.org/proceedings/2013/HT/PharmaSUG-2013-HT02.pdf.
- Clinical Data Interchange Standards Consortium (CDISC). 2013. *CDISC SDTM Implementation Guide*. Version 3.2. (November 26, 2013)
- Fan, Jueru. 2016. “Trivial Date Tasks? PROC FCMP Can Help.” *Proceedings of the Pharmaceutical SAS Users Group (PharmaSUG) 2016 Conference*. Cary, NC: SAS Institute Inc. Available at http://www.pharmasug.org/proceedings/2016/QT/PharmaSUG-2016-QT08.pdf.
- Food and Drug Administration (FDA), Center for Drug Evaluation and Research (CDER), 2011. “CDER Common Data Standards Issues Document.” Version 1.1 (December 2011). http://www.fda.gov/downloads/Drugs/DevelopmentApprovalProcess/FormsSubmissionRequirements/ElectronicSubmissions/UCM254113.pdf
- SAS Institute Inc. 2013. *Base SAS® 9.4 Procedures Guide*. Cary, NC: SAS Institute Inc. Chapters 22 and 23. Available at http://support.sas.com/documentation/cdl/en/proc/68954/PDF/default/proc.pdf.
- Scocca, David. 2013. “Developing Your SDTM Programming Toolkit.” *Proceedings of the SAS Global Forum 2013*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings13/178-2013.pdf.

SUGGESTED READINGS

Scocca (2013) discusses several tasks involved in programming SDTM data sets, and includes a macro function that converts partial dates without imputation.

Adams (2010) and Fan (2016) have examples of PROC FCMP functions useful for programming ADaM (Analysis Data Model) data sets, including functions that impute partial dates.

ACKNOWLEDGMENTS

The author gratefully acknowledges the following individuals for their help: Hunter Vega provided feedback on drafts of the manuscript and tested the function in the main example. Habtamu Benecha aided in running code on different versions of SAS.

CONTACT INFORMATION

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

Name: Noory Kim
Enterprise: CROS NT LLC
Address: 501 Eastowne Drive
City, State ZIP: Chapel Hill, NC 27514
Work Phone: 919-929-5015
Fax: 919-928-9320
E-mail: noory dot kim at crosnt dot com
Web: http://crosnt.com