Paper CC22

A Macro for Transforming Almost Any Character Calendar Date into a SAS Date Value

Ruben Chiflikyan, Mila Chiflikyan, and Donna Medeiros
RTI International, Research Triangle Park, NC

ABSTRACT

We provide a macro for the creation of SAS date values from almost all standard and non-standard types of character representations of Gregorian calendar dates. The program can work with quite loosely written 'contaminated' data, with, practically, no restrictions on the type and number of delimiters and sequence of month, day, and year values in a string to be transformed, e.g., "Jan 17th 2008", "2000-** Jan 2nd", "03##Sept.$$$02", "6/XII/08", " 2 012003", etc. The current version of the program supports the following date types: 1) month, day, and year are 'numbers', 2) year and month are 'numbers', 3) all types of Julian dates, 4) all standard week date formats, 5) all standard year-quarter combination, 6) day and year are 'numbers', while month is presented in Gregorian month names, 7) day and year are 'numbers', while month is presented as a Roman numeral, 8) special type of non-standard dates, where month, day, and year are 'numbers' but there are 0,1, or 2 delimiters. Each date type is transformed in a separate macro designed specifically for it. The usage of the program requires the user to provide laconic directives for identifying the type of date and the sequence of month, day, and year in the string that has a non-ambiguous number of high quality calendar date elements. The program is constructed in a modular fashion, which provides the user ease and flexibility when creating and adding new modules or updating and/or removing the existing ones. Even users with minimal knowledge of SAS can use it effectively.

INTRODUCTION

The creation of high quality data, including calendar dates as the valid SAS dates, is one of the important steps in the data preparation process [1]. A date in a calendar is a reference to a particular day represented within a calendar system. In most calendar systems, the date consist s of three parts: the day of month, month, and a year. There may be also additional parts, such as the day of the week. (See [2] and references therein to review calendar dates). A SAS date value is an integer, whose value is the number of days before or after January 1, 1960. In many cases, the SAS System provides significant number of date INFORMATS for reading raw calendar dates. Functionally, a date INFORMAT can be considered as an instruction given to the SAS System on how to transform specific character fields into a SAS date value. All known date INFORMATS are documented in the SAS Language Reference, Version 9. (See, also [3] for a handy review of all standard date INFORMATS, FORMATS, and date functions). However, in many cases, the imported raw dates can be encountered in a multitude of other forms that cannot be handled by the set standard date INFORMATS. Here are some samples of character dates in non-standard form: 'Jan 17th 2005', ' 2005-Jan 2nd', '03/./Sept.\.\02', '6/XII/2008', '2 012003', etc.. It could be caused by many reasons, including specificity of external dates created by many software packages that are not necessarily in the form recognizable by standard SAS INFORMATS. Additionally, the raw dates could be 'contaminated', i.e., could contain unwanted special symbols, spaces, etc., which could be originated due to various types defects when reading data from external sources. So, providing high quality calendar dates as valid SAS dates in timely and cost-effective fashion could be a quite challenging problem. To handle these situations, the programmer has to create his/her own technique to manipulate each new type of the original string individually. Many approaches and techniques have been developed and successfully implemented to handle various aspects of this multifaceted complex problem (see, e.g., [4, 5] and references therein). In particular, in [4], the author considers various situations on handling some of the dates nor recognized by the date INFORMATS supplied by SAS. In particular, the author emphasizing the point that INFORMATS have not been given the same enhancements that the formats have received, express the wish to have the INFORMAT analog of the picture format directives. A set of useful techniques when working with external dates are discussed in [5].

To overcome some of the problems mentioned above, we developed an approach that allows easily transform almost any practically interesting external character dates, which could be 'contaminated' and have quite loose formats, to valid SAS dates. To do it, the user, needs to provide easily created terse directives with information on the sequence of calendar dates elements (e.g., month, day, and year) for any of the 8 general types of dates considered in this paper. We assume that each date field in the dataset should be associated with a corresponding directive. This is a simple procedure and requires minimal SAS experience from user side. It is important to stress that the creation of directives is the *only* action from the side of the user, while the system takes care of the date type by itself. No

preliminary cleaning is required. The only precaution that should be taken is having non-ambiguous set of calendar elements in a raw date string.

The current version of the program supports the following types of dates: 1) month, day, and year are numbers; 2) year and moths are numbers; 3) all types of Julian dates; 4) all standard week date formats; 5) all standard year-quarter combination; 6) day and year are numbers, while month is presented in Gregorian month names; 7) day and year are numbers, while month is presented in Roman numerals; 8) special type of non-standard dates where month, day, and year are numbers, but there is only one delimiter.

The program is constructed using the modular approach, where each type of date, mentioned above, is transformed separately by appropriate macro designed for this specific date type only. It provides the user easiness and flexibility easily to create and add new modules or update and/or remove the already created ones. The macro constructed in the way that even the user with minimal knowledge of SAS can effectively use it.

The current version does not cover International dates as well as dates in a packed (binary) form. The modular structure of the program allows adding them easily in case of necessity.

THE FUNCTIONALITY OF THE PROGRAM

To understand how the program works, we provide below a simple example of program usage.

Let us assume that the researcher works with data provided by 3 agencies. In the raw data the date variable is a string variable, whose specific form changes from agency to agency. It is important to note that the program can work with contaminated data that could be in very loose format. Usually, one needs to clean the data and then create SAS date. In frames of this approach, in many practical cases the researcher does not have to do the cleaning. The program does all the necessary actions by itself. The only required condition is that the date elements that will be used in construction of the SAS date should be in not ambiguous form. If the date elements are in non ambiguous form, the type and specific order of special symbols that contaminate the data will be unimportant. This will be demonstrated in full scale later in the program. Here, to provide the general idea of this approach and clarify how the program works, we restrict ourselves with the simple demonstration below.

Say, the researcher needs to concatenate data that arrive from various states. The problem is that the form of date variable differs from state to another. Say, one state provides dates in “Nov 25, 2007”, the second in “October 1rst 09” form, the third one in the “ 09-19-03” form, etc.. Besides, because of the various reasons date fields could be contaminated. So, using the same date strings, their contaminated analogs could have the following form: “Nov**** 25, &^2007”, “Octob #@ er 1rst 09,,,,”, and “ 09-as19b!!-03^^^”. Regular way of handling of this type of dates requires mandatory data cleaning. It is important to note that the date field should contain non-ambiguous data. It means that you always have to have the necessary number of date elements needed for creation of SAS date. Afterwards, if the date is reduced to one of the standard form, the regular SAS INFORMATS can be applied. If the date string cannot be reduced to a one of the known forms that have INFORMATS, one needs to create SAS dates programmatically. Both the process of cleaning and creating of INFORMATS for nonstandard data require certain level of SAS experience. We propose an approach that simplifies significantly the creation of SAS dates requiring only a minimal intervention from the side of a user. The user needs only to provide terse instructions describing the date type.

Say, the first agency provided the text file that has the following form:

Agency	ID	Date
A	A001	2*/15/$$06
A	A002	2/#12/!1999
A	A003	1/1/ 00

Figure1. The original dataset from agency A.

Here, the first and second columns are agency’s name and identificator, correspondingly. The third column, Date, has contaminated dates in the form (day, month, year) form, which specifically in this case has dd/mm/yy form. As one can see, the dates are contaminated by a certain numbers of special symbols *, #, $, !, and spaces.

The second agency provides the data in the following form:

Agency	ID	Date
B	B001	01! sep 03
B	B002	03^ oct* 01
B	B003	02 $jul-- 03

Figure 2. The original dataset from agency B.

Here, you can see that the date composite elements (day, month, year) are presented in the ddMONyy form, which also is contaminated with !, ^, *, $, -, and spaces.

To create the analysis file, we need to concatenate both datasets and create date_new numeric variable that accepts SAS date values.

Our approach requires creation of a new variable that will provide terse directives - information about the date type to be analyzed and the sequence of month, day, and year values. This operation is done for each for each dataset separately. Since it is assumed that the dates within each agency have the same structure, the directive variable can be written only once and then copied to each observation using RETAIN option. Another important reason to have directives is to get rid of potential ambiguity when the form of the date (say, 01/02/03) can not be interpreted correctly without additional instructions.

As was mentioned above, the date string should contain non-ambiguous sequence of date elements, what is true, also, when using the standard INFORMATS. E.g., here are samples of ambiguous data: “01/03/04/05”, “Sep Dec 23 1987”, “14, 14, 2008”, etc.. The beauty of this approach is that the user are not worried about the specific type of contaminations on condition that date elements are non-ambiguous and correct (e.g., 1<= month <=12). The program analyzes the string and retrieves all necessary elements by itself.

We add a new column named ‘directive’ to the dataset created from the data provided by agency A.

Agency	ID	Date	directive
A	A001	2*/15/$$06	**nmdy**
A	A002	2/#12/!1999	**nmdy**
A	A003	1/1/ 00	**nmdy**

Figure 3. The original dataset from agency A with the added directive variable.

In doing it, we assume that the date type remains the same for all observations received from a same agency, i.e., the raw date cannot be 2/15/06 and Sep. 26, 2006 for any two observations. In this specific case date has mm/dd/yy(yy) form. The directive variable has the value ‘nmdy’, which is a terse description of the date type. Here, the ‘n’ character indicates that this specific date type will be handled by a specific macro macro, defined below. The ‘mdy’ string shows the exact sequence of month (m), day (d), and year (y) variables in date variable. It is important to that the special symbols, which represent contamination, can have arbitrary positions within the date string for dates even within one agency. The user does not to go into the details of the specific date representation

macro that handles the date string that have month information expressed as a name (Feb, February, FEBR, etc.), and the ‘dmy’ string provides information about the sequence day, month, and year in the string. Even the wrong spelling in month names is acceptable as long as base is intact, e.g., February, febroray, fEb., chfeB, etc., are all acceptable month names.

The original dataset from B agency with the added new variable is shown below.

Agency	ID	Date	directive
B	B001	01! sep 03	**xdmy**
B	B002	03^ oct* 01	**xdmy**
B	B003	02 $jul-- 03	**xdmy**

Figure 4. The original dataset from agency B with the added directive variable.

After concatenating datasets shown in Fig.3 and Fig.4, we obtain the dataset one having the form

Agency	ID	Date	directive
A	A001	2*/15/$$06	**nmdy**
A	A002	2/#12/!1999	**nmdy**
A	A003	1/1/ 00	**nmdy**
B	B001	01! sep 03	**xdmy**
B	B002	03^ oct* 01	**xdmy**
B	B003	02 $jul-- 03	**xdmy**

Figure 5. The concatenated datasets from the A and B agencies with the added directive variable.

Now, we call the almost(`dsin = , dsout = , str = , add = , sasdt =`) to create SAS date. Here, `dsin` and `dsout` are input and output datasets names, `str` contains value of the character raw date, `add` is the name of the INFORMAT created manually by user, and `sasdt` is the value of the SAS date. In this case macro should be called as follows: %almost(`dsin` = one, `dsout` = two, `str =` date, `add =` directive, `sasdt =` sasnum);

Agency	ID	date	directive	sasnum
A	A001	2*/15/$$06	**nmdy**	16847
A	A002	2/#12/!1999	**nmdy**	14287
A	A003	1/1/ 00	**nmdy**	14610
B	B001	01! sep 03	**xdmy**	15949
B	B002	03^ oct* 01	**xdmy**	15251
B	B003	02 $jul-- 03	**xdmy**	15888

Figure 6. The SAS date value added to the concatenated dataset by a designated macro.

USED INFORMATS' TYPES

Currently, the program handles the following types dates, which cover a lot of practical cases.

The '**nmdy**', '**nmyd', 'ndmy', 'ndym', 'nvyd', 'nvdy**' informats (used above) is allocated for any combination of month, day, and year, where their values are expressed as numbers. The character 'n' takes care of it. Here are some examples: '09:12:02', '09.12.02', 'dfs 09df/12 dh/02 dgdfgf'– contaminated, '09/12/02 ddd' – contaminated, ' ******** 09 -12qqe-02 db' –contaminated, ' 9 12 2', '10203', '010203', '20030102', '20030201', ' 01200302', '02200301', etc.

The **'omy'**, and **'oym'** informats for any month and year combination. The character 'o' is the informat name. Here are some examples: '02-2003','02-03', ' 2003.02', '03:02', '2003M02', '03M02', 'fff0203@#$' -contaminated, 'g02ww2003….' - contminated, '02M2003, etc.

The **'xmdy', 'xmyd', 'xdmy', 'xdym', 'xymd',** and **'xydm'** for any combination of month, day, and year, where the value of the month contains month name (worddate informat). Here are some examples: 'Jan 17th 2005', 'Jan# 24rth ^^2005' - contaminated, 'Jan 21st 2005','Jan 22nd 2005','Jan 23rd 2005', '23rd 2005 January', '2005, January 23rd', 'Jan 17.** 2005 -contaminated', 'Jan 17, 2005', 'January 17, 2005', 'Aug. 2nd, 2005', '17th January, 2005', '17th March, 2005', '9-May-2001', '02 SEP 03', '02 SEP 03', ' SEP 02 03', 'SEP 02 03', 'SEP 2 3', 'SEP 2 3', '02 03 SEP', '02 03 SEP', '9-sep-2001', '9-sep2001', '9-september-2001'.

The **'wkV','vkW', 'wkU'** weekdate informats. Note that wku, wkv, and wkw as values of &add.

Note that 'wkU' informat assumes that the week begins with Sunday, day of week=1, 'wkW' informat assumes that the week begins with Monday, day of week=1, and 'wkV' informat assumes that the week begins with Monday, day of week=1, and Jan 4th and first Thursday of year are included. Here are examples: 1969-W09-02, 1969W0902, 69W0902, 69W09, W09.

The **'jul'** julian date, Examples: 2003021, 2003-021, 03-021, 03021. Contaminations could be present.

The **'qry'** for year quarter combination. Here are examples: '2006^^&Q01' -contaminated, '06Q01', ' 2006Q1', etc.

The **'rmdy'**, **'rmyd'**, **'rdmy'**, **'rdym'**, **'rymd'**, and **'rydm'**, cover all cases where month is presented in the form of Roman characters: I=January, V=May, X=October, etc. Here are examples: '8 .I.1998', '8.II. 1998', '8.IV.1998', '1996.6.IV', 'III.1998.8', '8. *** VII.1998', etc..

To handle various forms of nonstandard data, such as in this case, the user needs to provide more details regarding the positions of month, year, and day within a string. For example, the informat **'m2**d1?!y4**' is used for dates of 01**2?!2003 type **'m2**d1y4'** while informat is used for 01**22003' type. Here is the list of pairs of informats and the corresponding dates: **'m2d1**y4'** for 012**2003, **'m2**y4d1'** for 01**20032, **'m2y4**d1'** for 012003**2, **'d1**m2y4'** for 2**012003, **'d1y4**m2' for** 22003**01, **'d1m2y4'** for 2012003, etc. Here, the '**y4**m2d1'** template has the following structure: 'y4' shows that the y is presented as a 4 digit number, then come 2 contaminated fields, then comes month presented as 2 digit, and finally, comes day as 1 digit (there is no preceding zero if days <=9).

THE PROGRAM

The program is constructed in the modular fashion. Each type of date, defined above, is transformed by the appropriate macro. Since the size of the full program exceeds the limits of the current paper, we restricted ourselves by presenting only `%mundane, %week date, %word date, and %quadrate macros.` The full electronic version of the program, including additionally, `%NMS date, %update, %nitrate, and %remediate` macros is available. Below you can find the program description. Please, note the first 4 macros are located *after* the main driving program. It is done with the goal not to overcrowd the paper. Technically, macros should be compiled before the main run.

The program is based on intensive use of PRX functions. Note, however, that because of space constraints, we are unable to provide all necessary details. The reader is advised to reference many SAS papers of the PRX functions. (For basics on PRX functions see, e.g., [6] and references therein.)

We provide here minimum information on PRX functions used in the four macros below.

The PRXPARSE function is used to create a regular expression, and it is placed in the data step at _N_=1. It will be used later by other PERL regular expression functions. It is executed once and its return value is retained.

The PRXMATCH function locates the position in a string, defined earlier by PRXPARSE function. The function returns the first position in a string described by regular expression.

The CALL PRXPOSN function returns the position and length for a capture buffer (a sub-expression defined in the regular expression).

The CALL PRXSUBSTR is used in conjunction with PRXPARSE function to locate the starting position and length of a pattern within a string. It returns the length of the match as well as the starting position.

The following PERL meta-characters used in this paper: \D matches a non-digit, \d matches a digit 0 to 9, {n} matches the previous sub-expression n times, {n, m} matches at least n and at most m occurrences of the preceding pattern, ? matches exactly 0 or 1 occurrences of the preceding pattern, * matches the previous sub-expression zero or more times, + matches the previous sub-expression one or more times.

DRIVER

Note that to run the driver one needs run all macros and initialize all macro variables below. We created several macro variables that contain delimiters used in raw dates. Note that the users can always modify these strings according to their specific needs.

```
%let YEARCUTOFF=1940; * yearcutoff as a macro variable;
options YEARCUTOFF=&YEARCUTOFF;
%let brd= _ ^ ! ? | *  % ~ ( )  , ; * date borders indicators;
%let mnt=JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC; * month base names;
%let delm='-',':',' ','/','.','_'; *; regular delimiters between values in %numdate macro;
%let del=. * \- : # + = _ \\ \/ ~ ^ ; * special symbols used as contaminations;
%let rom=*I* *II* *III* *IV* *V* *VI* *VII* *VIII* *IX* *X* *XI* *XII*; * Roman literals;
```

Below are the macro variables that contain temporary dataset variables that should be deleted after all manipulations.

```
%let roma = ptrom1 ptrom2 ptrom3 match1 match2 match3 pos1
lng1 pos2 lng2 pos3 lng3 found ss monr;     *from %romadate;
%let nstr = first match1 pos2 lng2 text addn del1 del2 m_num d_num y_num
num_m num_d num_y txt1 temps txt2 txt3 temps1 temps2 del_num;    * from %nstrdate;
%let quar = match1 match2 pos lng; *from %quardate;
%let num = match1 match2 match3 match4 pos1 lng1 pos2 lng2 pos3 lng3
str1 str2 str3 start2 length2 start3 length3 start4 length4; *from %numdate;
%let nms = match1 match2 match3 match4 pos1 lng1 pos2 lng2
pos3 lng3 pos4 lng4 start3 length3 str1 str2 start4 length4;   * from %nmsdate;
%let jul = match1 match2 match3 match4 temp1 temp2 low_b high_b  temp3 temp4;
%let week = match1 match2 match3 match4 match5 dt; *from %juldate;
%let word = match1 match2 match3 m d y;    *from %worddate;
%let initin = ptrd1 ptrn1 ptrn2 ptrn3 ptrn4 ptrs1  ptrs2 ptrs3 ptrs4
ptrj1 ptrj2 ptrj3 ptrj4 ptrw1 ptrw2 ptrw3 ptrw4 ptrw5
ptrwr1 ptrwr2  ptrwr3 ptrq1 ptrq2  ptrom1 ptrom2 ptrom3 string;    * from %init;
```

Here is the main macro that calls all the other macros, corresponding to each specific type of INFORMATS created by user. Note that the user can change the number of active macros depending on specific type of dates. Even new macro(s) can be added by user if special type dates will encounter, which will be not covered by existing ones.

```
%macro almost( dsin = , dsout = , str = , add = , sasdt = );
data &dsout(drop= &initin &word &week &jul &nms &num &quar &nstr &roma);
set &dsin;
%init;
if &add ^=' ' then
do; found = 0;
  string = upcase(' '||&str); *adding empty space;
  &add = upcase(trim(left(&add)));
  %numdate; %nmsdate; %juldate; %weekdate;
  %worddate; %quardate; %nstrdate; %romadate; end;
run;
%mend almost;

%almost( dsin = prx00,dsout = prx10,str = chardat, add = infrmt, sasdt = sasnum);
```

Here, `dsin` is the input dataset name, `dsout` is the output dataset name, `str` is the name of the variable containing the name of the raw date, `add` is the name of the informat that contains informat types created by user, `sasdt` is the name of variable that will contain SAS date values created from raw date.

INITIALIZATION OF THE PRX FUNCTIONS

Firstly, we initialize PRX functions to be used in the %init macro Note that we initialize variables for all 8 types of INFORMATS.

```
%macro init;
if _N_ = 1 then
do;
   * parse expression for each type of the delimiters in non-standard format;
   ptrd1 = PRXPARSE("/(\D+)(\d{1,4}\D*\d{1,4}\D*\d{1,4})(\D+)/");

  * parse expression for each type of the delimiters in mdy format;
   ptrn1 = PRXPARSE("/\D+(\d{1,4})\D+(\d{1,4})\D+(\d{1,4})\D+/");
   ptrn2 = PRXPARSE("/\D+(\d{5})\D+/");
   ptrn3 = PRXPARSE("/\D+(\d{6})\D+/");
   ptrn4 = PRXPARSE("/\D+(\d{8})\D+/");

    *  parse expression for each type of the delimiters in 'month, year'  format;
   ptrs1= PRXPARSE("/\D+(\d{2,4})\D+(\d{2,4})\D+/");
   ptrs2 = PRXPARSE("/\D+(\d{2,4})M(\d{2,4})\D+/");
   ptrs3 = PRXPARSE("/\D+\d{4}\D+/");
   ptrs4 = PRXPARSE("/\D+\d{6}\D+/");

    * parse expression for each type of the delimiters in Julian date format;
   ptrj1 = PRXPARSE("/\D+(\d{4})(\d{3})\D+/");
   ptrj2 = PRXPARSE("/\D+(\d{4})\D+(\d{3})\D+/");
   ptrj3 = PRXPARSE("/\D+(\d{2})(\d{3})\D+/");
   ptrj4 = PRXPARSE("/\D+(\d{2})\D+(\d{3})\D+/");

 * parse expression for Weekdate format;
   ptrw1 = PRXPARSE("/\D+\d{4}-W\d{2}-\d{2}\D+/");
   ptrw2 = PRXPARSE("/\D+\d{4}W\d{4}\D+/");
   ptrw3 = PRXPARSE("/\D+\d{2}W\d{4}\D+/");
   ptrw4 = PRXPARSE("/\D+\d{2}W\d{2}\D+/");
   ptrw5 = PRXPARSE("/[&brd]W\d{2}\D+/");

* parse expression for each type of the delimiters in Worddate format;
   ptrwr1 = PRXPARSE("/[&brd](&mnt).+(\d{1,4})\D+(\d{1,4})\D+/");
   ptrwr2 = PRXPARSE("/\D+(\d{1,4})\D+(&mnt).+(\d{1,4})\D+/");
   ptrwr3 =PRXPARSE("/\D+(\d{1,4})\D+(\d{1,4})\D+(&mnt).+[&brd]/");

* parse expression for year-quarter combination;
   ptrq1 = PRXPARSE("/\D+\d\d\d\dQ\d{1,2}\D+/");
   ptrq2 =PRXPARSE("/\D+\d\dQ\d{1,2}\D+/");

* parse expression for each type of the delimiters when month is in roman format;
ptrom1 =
PRXPARSE("/[&del]+(\d{1,4})[&del]+(I*X*I*I*I*V*I*I*I*)[&del]+(\d{1,4})[&del]+/");
ptrom2 = PRXPARSE("/[&del]+(I*X*I*I*I*V*I*I*I*)[&del]+(\d{1,4})\D+(\d{1,4})[&del]+/");
ptrom3 =
PRXPARSE("/[&del]+(\d{1,4})[&del]+(\d{1,4})[&del]+(I*X*I*I*I*V*I*I*I*)[&del]+/");
RETAIN ptrw1 ptrw2 ptrw3 ptrw4 ptrw5 ptrwr1 ptrwr2  ptrwr3 ptrq1 ptrq2
   ptrom1 ptrom2 ptrom3 ptrj1 ptrj2 ptrj3 ptrj4 ptrs1 ptrs2 ptrs3 ptrs4
   ptrn1 ptrn2 ptrn3 ptrn4 ptrd1;
end;
%mend init;
```

%WORDDATE MACRO

It is used for contaminated dates of all 6 combinations of day, month, and year types, where month is presented by its base name, e.g., ’Jan @@!17, 2005’, ‘*January* 17*, *2005*’, ‘Aug. 2nd, 2005’, ‘17th January, 2005’, etc.

```
%macro worddate;
```

```
%global word;
%let word = match1 match2 match3 m d y;
if index(&add, 'X')>0 then
do;
  match1 = PRXMATCH(ptrwr1, string);
  match2 = PRXMATCH(ptrwr2, string);
  match3 = PRXMATCH(ptrwr3, string);

  if (match1 gt 0)  then
  do;
    call prxposn(ptrwr1, 1, pos1, lng1);
    call prxposn(ptrwr1, 2, pos2, lng2);
    call prxposn(ptrwr1, 3, pos3, lng3);
    if index(&add, 'XMDY') > 0 then
    do;
       m = substr(string, pos1, lng1);
      d = substr(string, pos2-1, lng2+1);
      y = substr(string, pos3, lng3);
   end;
    else if index(&add, 'XMYD') > 0 then
    do;
       m = substr(string, pos1, lng1);
      y = substr(string, pos2, lng2);
      d = substr(string, pos3, lng3);
    end;

    &sasdt. = input(compress(d||m||y), date10.);
     &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
     found = 1;
  end;

  else if (match2 gt 0) then
  do;
    call prxposn(ptrwr2, 1, pos1, lng1);
    call prxposn(ptrwr2, 2, pos2, lng2);
    call prxposn(ptrwr2, 3, pos3, lng3);
    if index(&add, 'XDMY') > 0 then
    do;
      m = substr(string, pos2, lng2);
      d = substr(string, pos1-1, lng1+1);
      y = substr(string, pos3, lng3);
    end;
    else if index(&add, 'XYMD') > 0 then
   do;
       m = substr(string, pos2, lng2);
      y = substr (string, pos1, lng1);
      d = substr(string, pos3-1, lng3+1);
   end;

    &sasdt.= input(compress(d||m||y), date10.);
     &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
     found = 1;
  end;

  else if (match3 gt 0) then
  do;
    call prxposn(ptrwr3, 1, pos1, lng1);
    call prxposn(ptrwr3, 2, pos2, lng2);
    call prxposn(ptrwr3, 3, pos3, lng3);
    if index(&add, 'XDYM') > 0 then
    do;
```

```
      m = substr(string, pos3, lng3);
      d = substr(string, pos1, lng1);
      y = substr(string, pos2, lng2);
    end;
    else if index(&add, 'XYDM') > 0 then
    do;
      m = substr(string, pos3, lng3);
      y = substr(string, pos1, lng1);
      d = substr(string, pos2, lng2);
    end;

    &sasdt. = input(compress(d||m||y), date10.);
    &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
     found = 1;
  end;
  return;
end;
%mend worddate;
```

%WEEKDATE MACRO

It used for working with contaminated dates having **'wkV', 'vkW', 'wkU'** INFORMATS. Here are examples: '1969-W09-02','1969 **W09&02', '69 W09 02', '*69 W 09*', 'W @@09'.

```
%macro weekdate;
if index(&add, 'WKU') > 0 or index(&add, 'WKV') > 0 or index(&add, 'WKW') > 0 then
do;
  match1 = PRXMATCH(ptrw1, string);
  match2 = PRXMATCH(ptrw2, string);
  match3 = PRXMATCH(ptrw3, string);
  match4 = PRXMATCH(ptrw4, string);
  match5 = PRXMATCH(ptrw5, string);

       if match1 > 0 then  call prxsubstr(ptrw1, string, pos, lng);
  else if match2 > 0 then  call prxsubstr(ptrw2, string, pos, lng);
  else if match3 > 0 then  call prxsubstr(ptrw3, string, pos, lng);
  else if match4 > 0 then  call prxsubstr(ptrw4, string, pos, lng);
  else if match5 > 0 then  call prxsubstr(ptrw5, string, pos, lng);

  if sum(match1, match2, match3, match4, match5) > 0 then
  do;
    dt=trim(left(substr(string, pos+1, lng-1)));
         if index(upcase(&add), 'WKV') > 0 then &sasdt. = input(dt,weekv11.);
    else if index(upcase(&add), 'WKU') > 0 then &sasdt. = input(dt,weeku11.);
    else if index(upcase(&add), 'WKW') > 0 then &sasdt. = input(dt,weekw11.);
    &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
    found = 1;
  end;
  return;
end;
%mend weekdate;
```

%NUMDATE MACRO

It used to work with contaminated dates expressed in any of 6 combinations of month, day, and year components, all of them are expressed as numbers. Here are some examples: '09/12/02', '09-12-02', 'dfs 09df/12 dh/02 dg' '09///12////02 dcondd'' ***** 09 ?? -12qqe-02 db', '10203', '20030102', '20030201', ' 01200302', '02200301', etc..

```
%macro numdate;
if index(&add, 'N') > 0 then
do;
  match1 = PRXMATCH(ptrn1, string);
  match2 = PRXMATCH(ptrn2, string);
  match3 = PRXMATCH(ptrn3, string);
  match4 = PRXMATCH(ptrn4, string);

  if match1 gt 0 then
  do;
    call prxposn(ptrn1, 1, pos1, lng1);
    call prxposn(ptrn1, 2, pos2, lng2);
    call prxposn(ptrn1, 3, pos3, lng3);
    str1 = substr(string, pos1, lng1);
    str2 = substr(string, pos2, lng2);
    str3 = substr(string, pos3, lng3);
    found = 1;
  end;

  else if match2 gt 0 then
  do;
    call prxsubstr(ptrn2, string, start2, length2);
    str1 = substr(string, start2+1, 1);
    str2 = substr(string, start2+2, 2);
    str3 = substr(string, start2+4, 2);
    found = 1;
  end;

  else if match3 gt 0 then
  do;
    call prxsubstr(ptrn3, string, start3, length3);
    str1 = substr(string, start3+1, 2);
    str2 = substr(string, start3+3, 2);
    str3 = substr(string, start3+5, 2);
    found=1;
  end;

  else if match4 gt 0 then
  do;
    call prxsubstr(ptrn4, string, start4, length4);
    if index(&add, 'NYMD') > 0 or index(&add, 'NYDM') > 0 then
    do;
      str1 = substr(string, start4+1, 4);
      str2 = substr(string, start4+5, 2);
      str3 = substr(string, start4+7, 2);
      found = 1;
    end;
    else if index(&add, 'NMYD') > 0 or index(&add, 'NDYM') > 0 then
    do;
      str1 = substr(string, start4+1, 2);
      str2 = substr(string, start4+3, 4);
      str3 = substr(string, start4+7, 2);
      found = 1;
    end;
    else if index(&add, 'NMDY') > 0 or index(&add, 'NDMY') > 0 then
    do;
      str1 = substr(string, start4+1, 2);
      str2 = substr(string, start4+3, 2);
      str3 = substr(string, start4+5, 4);
      found = 1;
    end;
  end;
```

```
  if found = 1 then
  do;
         if index(&add, 'NMDY') then do; m = str1; d = str2; y = str3; end;
    else if index(&add, 'NMYD') then do; m = str1; d = str3; y = str2; end;
    else if index(&add, 'NDMY') then do; m = str2; d = str1; y = str3; end;
    else if index(&add, 'NDYM') then do; m = str3; d = str1; y = str2; end;
    else if index(&add, 'NYMD') then do; m = str2; d = str3; y = str1; end;
    else if index(&add, 'NYDM') then do; m = str3; d = str2; y = str1; end;

    &sasdt. = mdy(m, d, y);
    &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
  end;
  return;
end;
%mend numdate;
```

%QUARDATE MACRO

Used to work with contaminated dates presented in year-quarter combinations Here are some samples: '2006^^&Q01' , '06Q01', ' 2006Q1', etc.

```
%macro quardate;
if index(&add, 'YQ') > 0 then
do;
  match1 = PRXMATCH(ptrq1, string);
  match2 = PRXMATCH(ptrq2, string);
       if match1 > 0 then  call prxsubstr(ptrq1, string, pos, lng);
  else if match2 > 0 then  call prxsubstr(ptrq2, string, pos, lng);

  if sum(match1, match2)>0 then
  do;
    &sasdt. = input(trim(left(substr(string, pos+1, lng-1))),yyq9.);
    &sasdt._f = &sasdt.;
    format &sasdt._f mmddyy8.;
    found = 1;
  end;
  return;
end;
%mend quardate;
```

USAGE OF THE PROGRAM

To operate the current version of the program, the user needs to provide the following input parameters to the %almost macro: the input and output datasets' names, the variable name that contain the original raw data string, variable name that contains INFORMAT names created by the user, and the variable name, which will contain the calculated SAS dates values. The current version of the program uses temporary variable names, which can be found in `&initin, &word, &week, &jul, &nms, &num, &quar, &nstr, and &roma` macro variables. Note that the potential conflict could be happen in case if the names of the native dataset variables occasionally will be same as variables' names used in %almost. To solve this problem, one can rename the temporary variables in a macro with, e.g., such names as __varname1__,__varname2__, etc., what will allow distinguish them from the native variables having more traditional names. Another solution is to use the randomly generated sufficiently long variable names which will contain in their names almost unique combination numbers and letters. We decided to leave the program as it, because our main goal was a presentation and discussion of the principles upon of which the approach is based, and any modification of this type would impair the visual clarity of presentation. Nevertheless, in its present form, the program has been used many times by the authors on condition that all precautions on matching variables' names were made. On some occasions, we created a separate dataset, containing only IDs and raw date string(s), ran the %almost macro, created the corresponding SAS date for all observations, and then, merged back to the rest of data.

As one can see, %almost macro is very handy and is highly recommended to use when you are handling raw dates presented in different form each formats. This type of situation arises when one works with date supplied from many

sources, than contain dates in almost arbitrary form. As one can see, we use a lot of SAS functions the create overheads, and the processing time increases significantly. We have checked the programs efficiency for many types of raw dates for a dataset of 1,500,000 observations. According to our estimations, it has efficiency about 10,000 observations per second.

CONCLUSION

The created macro is very efficient when transforming character Gregorian calendar dates into SAS date values. It can handle almost all types of standard and non-standard date types used in practice. The strings contained raw dates can be in a very loose format and 'contaminated' with other special symbols not belonging to the date elements. To use the program, the user is required to provide a terse description of date, INFORMAT, which includes date type and the sequence of date elements. Note that no preliminary data cleaning is required. The program is based on the modular structure principle, and new modules could be easily added to the list already existed one. The program can be used easily by user with a minimum SAS knowledge. By using it, one can simplify significantly the process of creation of SAS dates from many practically interesting types of raw date strings, that otherwise could be a challenge requiring time and resources and concentrate instead on more creative tasks. The full electronic version of the code is available and can be requested from the authors.

REFERENCES

1. Morgan, Derek P. 2006, The Essentials Guide to SAS Dates and Times. Cary, NC: SAS Institute Inc.
2. Calendar date, Wikipedia, http://en.wikipedia.org/wiki/Calendar_date. 10/8/2008.
3. Rhoades, Steve, Wanna Date? NESUG 2006, Paper PO-13.
4. Chakravarthy, Venky, Have a Strange DATE? Create your own INFORMAT to Deal with Her. SUGI 27, Paper 101-27.
5. Droogendyk, Harry, In(Formats) (In)Decently Exposed. SESUG 2004, Paper IN06.
6. Cassel, David, The Basics of the PRX Functions, SAS Global Forum 2007, paper 223-2007.

ACKNOWLEDGMENTS

The authors wish to thank Jean E. Richardson (RTI) for many valuable suggestions and discussions and Joe Nofziger (RTI) for his comments, as well as and Paul Dorfman for some discussions.

CONTACT INFORMATION:

Ruben Chiflikyan,
Research Triangle Institute, International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work phone: (919) 541 – 6064
Fax: (919)541-6178
Email: rchiflikyan@rti.org
Web: http://www.rti.org

Mila Chiflikyan
Research Triangle Institute, International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work phone: (919) 541 – 6064
Fax: (919)541-6178
Email: rchiflikyan@rti.org
Web: http://www.rti.org

Donna Medeiros
Research Triangle Institute, International
3040 Cornwallis Road, PO Box 12194
Research Triangle Park, NC, 27709-2194
Work Phone: (919)6000-21064
Fax: (919)541-6178
E-mail: djm@rti.org
Web: http://www.rti.org