Paper 85-25

Generating Dates Automatically

Armando V. Fabia, Canadian Imperial Bank of Commerce, Toronto, CANADA

ABSTRACT

This paper illustrates how we can use the date functions, particularly INTNX, and the CALL SYMPUT routine to automatically produce date values. Applications include calling data sets stamped with dates, updating dates in a data step expression, and putting dates into reports.

INTRODUCTION

Dates may be used or processed in certain applications. If these dates are refreshed regularly to reflect current information, an automatic process to generate them must be in place. In this paper, we will examine some applications that use dates, propose an automated solution to refresh the dates, and then revisit the applications to make the required automation.

A basic knowledge of SAS programming is required to understand the text. In particular the reader should have an idea of macro processing, SAS date formats, put statements, and some functions, including TODAY(), UPCASE and LEFT.

SOME APPLICATIONS

Let us examine some SAS data sets and codes to motivate the discussion. The data sets below are stamped with dates in YYMMDD4. format.

LIBRARY	*MEMBER*
/project1/segment/cust/s0003	scr0003
/project1/segment/cust/s0004	scr0004

The first data set contains customer segmentation and scoring as of the end of March 2000. The second data set is as of end of April 2000.

The following job is submitted on April 2000. It reads the file scr0003 from the library /project1/segment/cust/s0003 and then uses the CPORT facility to transport the created file to another system.

```
Example 1.
libname dd1 '/project1/segment/cust/s0003';
libname dd2 'usr/sas/users/avf';

proc datasets dd=dd2;
delete scores;
run;

data dd2.scores;
set dd1.scr0003(keep=cust score segment);
run;

filename tranfile '/usr/sas/users/avf/scr';
proc cport data=dd2.scores file=tranfile
v608;
run;
```

Suppose this job is run on a monthly basis and that its input data set must be one month behind. Thus on May 2000, we update the input library and file names into /project1/segment/cust/s0004 and scr0004, respectively, before running the job again. There must be a way to deal with these changes such that we do not have to do them every month.

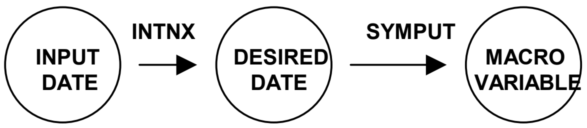
Here is another example. Example 2 is a statement that is part of an application submitted on April 2000. It attaches footnotes to a report depending on the value of the field DATAASOF.

```
Example 2.
IF DATAASOF='FEBRUARY 29, 2000' THEN
PUT // @2 'NOTES:'
    // @2 "(1) * = PORTFOLIO CLOSED AS OF
MARCH 31, 2000"
     / @2 "(2) TOTAL # OF CLOSED PORTFOLIOS:"
       @36 TPF;
ELSE
PUT // @2 'NOTES:'
    // @2 "(1) * = NEW PORTFOLIO"
     / @2 "(2) TOTAL # OF NEW PORTFOLIOS:"
       @36 TPF;
```

The application determines if a portfolio has been closed or is new based on data collected from the past two months. If it is to be submitted again on May 2000, the dates FEBRUARY 29, 2000 and MARCH 31, 2000 must first be changed to MARCH 31, 2000 and APRIL 30, 2000, respectively. If this application becomes a monthly process, we must find a way to deal with the dates.

PROPOSED SOLUTION

When applications such as these examples require regular date changes, automation is highly recommended. A good automated solution is to create a macro variable through the CALL SYMPUT routine whose value is a date produced by the INTNX function.



THE *INTNX* FUNCTION

Let us begin with the date function INTNX. It generates a SAS date value that is a given number of intervals from a starting value. Its syntax is

INTNX('*interval*',*from*,*number*).

In this paper, we illustrate the function using DAY, MONTH, and YEAR as arguments for *interval*. (There are other possible intervals.)

The argument *from* is the starting point and must be a SAS expression that gives a date value. If the starting point is the current date, we can use the date expressions TODAY() or its alias DATE().

The *number* is an integer that specifies the number of intervals from the starting value.

Let us look at the following statements.

```
date1 = intnx('year',date(),-2);
date2 = intnx('month',today(),-2);
date3 = intnx('day',today(),-2);
date4 = intnx('year',today(),0);
date5 = intnx('month',date(),0);
date6 = intnx('year',today(),1);
date7 = intnx('month',today(),1);
date8 = intnx('day',date(),1);
```

The INTNX function counts the number of intervals from fixed interval beginnings when the argument for *interval* is YEAR or MONTH. When the *interval* argument is YEAR, INTNX begins counting the number of years from January 1 of the year given in the *from* argument. Thus, if these statements are submitted on April 11, 2000, date1 will return a SAS date value equivalent to January 1, 1998.

When the *interval* argument is MONTH, INTNX begins counting the number of months from the first of the month given in the *from* argument. Thus date2 is equivalent to February 1, 2000.

If the *interval* argument is day, then INTNX simply counts the number of days from the date given in the *from* argument. Therefore date3 returns the equivalent of April 9, 2000.

The statements involving date4 and date5 produce SAS date values equivalent to January 1, 2000 and April 1, 2000, respectively.

The last three statements advance April 11, 2000 to January 1, 2001, May 1, 2000 and April 12, 2000, respectively.

Beginning Release 6.11, the INTNX syntax has been updated to the form

INTNX(‘*interval*’,*start-date*,*increment*<,’*alignment*’>).

It now allows us to advance dates to the beginning, the middle, or to the end of the interval. The first three arguments have the same definitions as in the initial syntax. The fourth optional argument *alignment* has three possible values:

BEGINNING (or aliases BEGIN, BEG, and B) which is the default.
MIDDLE (or aliases MID and M) which advances the date to the midpoint of the interval.
END (or alias E) which advances the date to the end of the interval.

The following statements, when run on April 11, 2000, return November 30, 1999 and May 16, 2000, respectively.

```
date9 = intnx('month',today(),-5,'end');
date10 = intnx('month',today(),1,'middle');
```

Take note that the midpoint of a month with 28 days is the 14th day; the midpoint of a month with 29 or 30 days is the 15th; and that of a month with 31 days is the 16th.

The next two statements return December 31, 1997 and July 2, 2003, respectively.

```
date11 = intnx('year',today(),-3,'end');
date12 = intnx('year',today(),3,'middle');
```

What dates are returned by the following statements? Date14 helps you find out the midpoint of a leap year.

```
date13 = intnx('month',today(),-2,'end');
date14 = intnx('year',today(),0,'middle');
```

To put these dates into a desired date format, we use the put function. The following statements produce 0003 and FEB2000, respectively.

```
date15 = put(intnx(‘month’,date(),-1),
yymmdd4.);
date16 = put(intnx(‘month’,today(),-2),
monyy7.);
```

THE *CALL SYMPUT* ROUTINE
To assign this information into a macro variable, we utilize the CALL SYMPUT routine with the syntax

CALL SYMPUT(*argument1*,*argument2*).

Argument1 identifies the macro variable and *argument2* identifies its value. The example

```
call symput(‘yrmth’,
put(intnx(‘month’,today(),-1),yymmdd4.));
```

when run this April creates the macro variable YRMTH whose value is 0003.

The proposed solution uses the SYMPUT routine to create a macro variable whose value is the desired date produced from a DATA _NULL_ step.

REVISING THE APPLICATIONS
We are now ready to revise the examples. Take note that in Example 1 the job is to be run on the current month using data of the previous month. The INTNX function helps us convert the current month to the previous month; that is,

```
intnx(‘month’, today(),-1).                    (1)
```

We then need to put (1) in a date format similar to what appears in the input data set name. Hence we write (1) in the yymmdd4. format as follows.

```
put(intnx(‘month’, today(),-1),yymmdd4.)) (2)
```

Finally, we must create a macro variable that takes the value produced in (2). This is achieved by the CALL SYMPUT routine. An automated solution to Example 1 is therefore given below.

```
Example 1 revised.
data _null_;
call symput(‘yrmth’,
put(intnx(‘month’, today(),-1),yymmdd4.));
run;

libname dd1 "/project1/segment/cust/s&yrmth";
libname dd2 '/usr/sas/users/avf';

proc datasets dd=dd2;
delete scores;
run;

data dd2.scores;
set dd1.scr&yrmth(keep=cust score segment);
run;

filename tranfile '/usr/sas/users/avf/scr';
proc cport data=dd2.scores file=tranfile
v608;
run;
```

To automate the date changes required of the second example, we have to create two macro variables for the dates in the “footnote” statement. Keep in mind that each date is the last day of each of the past two months. Two solutions are proposed. The first solution involves the old INTNX syntax and the alternative solution involves the new syntax. Let us examine the first one.

```
Example 2 revised.
DATA _NULL_;
MTH=INTNX('MONTH',TODAY(),0);
CALL SYMPUT('MTHN',UPCASE(LEFT
(PUT(INTNX('DAY',MTH,-1),WORDDATE18.))));
```

```
MTHF=INTNX('MONTH',TODAY(),-1);
CALL SYMPUT('MTHB',UPCASE(LEFT
(PUT(INTNX('DAY',MTHF,-1),WORDDATE18.))));
RUN;
:
:
IF DATAASOF="&MTHB" THEN
PUT // @2 'NOTES:'
     / @2 "(1) * = CLOSED PORTFOLIO AS OF
&MTHN"
     / @2 "(2) TOTAL # OF CLOSED PORTFOLIOS:"
       @36 TPF;
ELSE
PUT // @2 'NOTES:'
    // @2 "(1) * = NEW PORTFOLIO"
     / @2 "(2) TOTAL # OF NEW PORTFOLIOS:"
       @36 TPF;
```

In the first solution we create two variables (called MTH and MTHF) whose values are the beginning dates of the current month and the previous month. To get the end dates of the past two months, we simply subtract one day from MTH and MTHF.

The use of the UPCASE function, the LEFT function and the WORDDATE format is necessary to match the characteristics of the field DATAASOF and the layout of the report. The UPCASE function is used because SAS writes WORDDATE. formats in title case and the LEFT function is used because formatted SAS dates are right–aligned.

An alternative solution to Example 2, utilizing the *alignment* argument, is a shorter version of the first solution. Since we are interested in the last dates of the past two months, the proper *alignment* argument is END.

```
Example 2 revised (alternative solution).
DATA _NULL_;
CALL SYMPUT('MTHN',UPCASE(LEFT
(PUT(INTNX('MONTH',TODAY(),-1,'END'),
WORDDATE18.))));
CALL SYMPUT('MTHB',UPCASE(LEFT
(PUT(INTNX('MONTH',TODAY(),-2,'END'),
WORDDATE18.))));
RUN;
:
:
IF DATAASOF="&MTHB" THEN
PUT // @2 'NOTES:'
    // @2 "(1) * = CLOSED PORTFOLIO AS OF
&MTHN"
     / @2 "(2) TOTAL # OF CLOSED PORTFOLIOS:"
       @36 TPF;
ELSE
PUT // @2 'NOTES:'
    // @2 "(1) * = NEW PORTFOLIO"
     / @2 "(2) TOTAL # OF NEW PORTFOLIOS:"
@36 TPF;
```

A FINAL EXAMPLE

Let us consider one final example. This time it involves creating a new data set based on the values of a date field.

```
Example 3.
data out1;
set in1;
if "01MAR2000"d <= srvdate <= "31MAR2000"d;
run;
```

Suppose this DATA step is part of a monthly process that creates a new data whose values for the date field SRVDATE are only from the previous month. Since both beginning and end dates of the month are required, the *alignment* argument for the first of the month is BEGINNING and that for the end of the month is END.

Here is a revised version of Example 3. We actually do not need to specify the alignment for DAYF because BEGINNING is the default.

```
Example 3 revised.
data _null_;
call symput('dayf',
put(intnx('month',today(),-1,'beginning'),
date9.));
call symput('dayl',
put(intnx('month',today(),-1,'end'),
date9.));
run;

data out1;
set in1;
if "&dayf"d <= srvdate <= "&dayl"d;
run;
```

A Word of Caution

If you are planning to use the system–generated date SYSDATE, be sure to use a YEARCUTOFF= option. If you use the code

```
options yearcutoff=1950;

data _null_;
call symput('mthn',
(put(intnx('month',"&sysdate"d,-4),
worddate18.)));
run;
```

on April 11, 2000, it will give you December 1, 1999. Without the YEARCUTOFF, it returns December 1, 1899 because SAS recognizes “&sysdate”d as April 11, 1900.

CONCLUSION

We have seen three situations where dates need to be generated or updated. These situations have the common characteristics of being periodically run and of having dates that must reflect current data. In such cases, automatically generating or updating the dates has removed the manual aspect of changing the dates. The solution that is proposed is to use the INTNX function to generate the desired date and then to assign it to a macro variable through the CALL SYMPUT routine.

REFERENCES

Language Reference, BASE SAS Documentation.

CONTACT INFORMATION

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

Armando V. Fabia
Canadian Imperial Bank of Commerce (CIBC)
Commerce Court West 7th Floor
Toronto M5L 1A2 CANADA
Work Phone: (416) 980-4268
Fax: (416) 304-4931
Email: a_fabia@yahoo.com