Paper 1861-2018

Finding and Generalizing a "Best Before" Date

Marje Fecht, Prowerk Consulting

ABSTRACT

Do you need a “Best Before” date 18 months from manufacture, and represented as a month end date?
Do you manually provide dates as input to your processes?
Do you struggle to get dates into the right format for database queries, or for reports and dashboards?

This presentation will help you **find the right date**, and then **generalize the coding** to avoid manual input, repetitive and messy coding, and frustration.

Examples emphasize the easy manipulation of dates, and focus on generalization to support **flexible coding**, including:

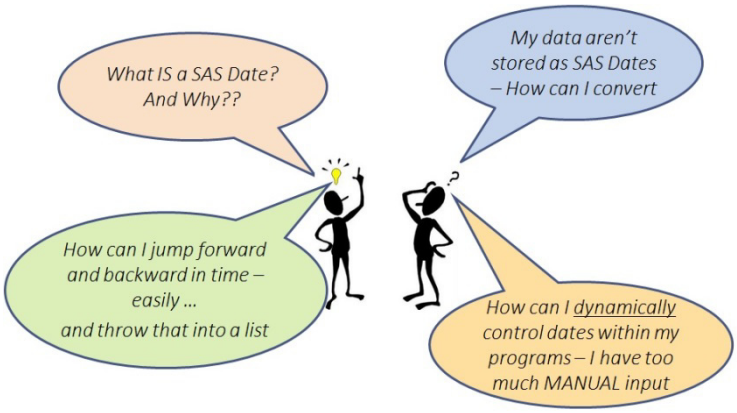
- Dynamically identifying date ranges, such as reporting and analytics periods (current calendar year; most recent 6 months; past 90 days; current fiscal year; year over year)
- Dynamically generating field names that represent date values or ranges
- Controlling the appearance of date values in reports
- Generating date-time stamps for file names, without special symbols.

INTRODUCTION

Are you an analyst who delivers insight for decision making? I would bet that at least 80% of what you deliver involves “past year performance”, or “month over month” comparisons, or some type of analytics that involves date/time periods. And, like many other SAS programmers, you probably deal with challenges like dates that are input as character strings, struggles with moving forward and backward in time (do you just add or subtract some quasi # of days?), and programs that require manual input to handle dates.

Can you guess what concept gets some of the highest traffic on support.sas.com???? **Dates**!!

Whether you are new to SAS, adding SAS to your vast toolbox, or a SAS user who has a lot of manual date processes, you may benefit from some of the tips and tricks in this paper.



SAS DATES . . . THE ORIGIN

SAS Date variables are numeric and stored as the INTEGER number of days since January 1, 1960 – both forward (positive) and backward (negative) in time. Thus, January 1, 1960 would be stored as 0. This methodology easily accommodates nuances in our calendar, such as leap year.

Limitations: Valid from 1582 AD to 19900 AD

Note that 1582 is the beginning of the Gregorian calendar.

Origin: For an interesting perspective, see Rick Langston's notes:

https://web.archive.org/web/20080706004217/http:/support.sas.com/community/newsletters/news/insider/dates.html

How do you determine Today's Date, as a SAS Date value?

Use either the **today()** or **date()** functions. They are equivalent , and parentheses are required since they are function calls. The function calls return the date ***at the time the function executes***.

```
Today_date = today();    /* # days today is since 1/1/1960 */

Today_date = Date();     /* # days today is since 1/1/1960 */
```

How do you find yesterday?

```
Yesterday = today() - 1;
```

SUBSET DATA BASED ON DATES

Suppose you need to subset your data for a specific date.

If there is a date you have in mind, you can use a **date constant**, and let SAS do the conversion.

```
where SaleDate = '23Mar2017'd;
```

Notes: Specify the date within single or double quotes, immediately followed by **d** or **D**.

Use the date7. Format (23Mar17) or date9. Format (23Mar2017).

If you want to subset **dynamically** using a general rule, then use date functions.

Example: Include all dates in **current** month and year, using the **month**, **year**, and **today** functions:

```
where month(SaleDate) = month( today() )
  and  year(SaleDate)  = year( today() );
```

NOW, LET'S CONSIDER EFFICIENCY AND PRACTICALITY

Suppose you are making use of the **today()** function throughout your program, and further suppose that you have a lot of data to process. The last few examples could generate A LOT of repeated calls to the `today()` function. If your program starts running at 11:55 PM and is still running at 12:15 AM, will you still be working with the same value of **today()** ?

NO!

How many times should you execute `today()` in your program / process?

ONE TIME, so that the value is constant for the entire execution!

By utilizing one call to `today()` and then making that value available to your entire process, you

- Improve efficiency (reduced function calls)
- Guarantee a single comparison date for processes that could cross midnight.

Create a macro variable, **today_dt**, that contains the value of today's date as a SAS date value (integer) at the time the `%let` executes.

```
%let Today_dt = %sysfunc( today() );
```

The previous example would now be coded as

```
where month(SaleDate) = month( &today_dt )
  and  year(SaleDate)  = year(&today_dt);
```

CREATE MONTH-END DATE VALUES

Thanks to our calendar, if we need to locate a Month End Date value, we can't just use the 30 or 31 or 28 or 29th of a month. It depends on the month. I have seen countless program examples that have month end dates being computed as the ***first day of the NEXT month minus 1***. Using SAS functions, there is a much cleaner solution.

Suppose we want to extract all observations for which the me_dt (month end date) on the observation is equal to the CURRENT month end date. The **INTNX** function is the solution.

```
/**** Extract all data for current month ***/
  If me_dt =
        INTNX ( 'MONTH'     /* increment = month */
              , today()     /* start at today */
              , 0           /* move ZERO months */
              , 'E')        /* return END of current month */
    ;
```

Result on 15Nov2017 is ME_DT = 21153 - *the SAS Date value corresponding to 30Nov2017*

The INTNX function moves forward and backward in time, based on the **INT**erval you specify, and is handy to dynamically create different variations of dates, since it *increments dates by intervals*.

INTNX (*interval*, *from*, *n* < , *alignment* > **) ;**

- ***interval*** **-** interval name eg: 'MONTH', 'DAY', 'YEAR' , etc
- ***from*** **-** a SAS date value (for date intervals) or datetime value (for datetime intervals)
- ***n*** **-** number of intervals to increment from the interval that contains the *from* value
  negative = backward in time
  positive = forward in time
  zero = same time interval
- ***alignment*** **-** alignment of resulting SAS date, within the interval.

  Eg: **B**EGINNING, **M**IDDLE, **E**ND, **S**AME.

DYNAMIC DATE VALUES

Functions give you the ability to dynamically generate date values. The above example returns a date that is the LAST day of the Current month.

How can I generate the first day of the month, for two months prior to today, and store it in a **SAS** variable?

```
Month_prev_2 =
          INTNX ( 'MONTH'    /* increment = month */
                , today()    /* start at today */
                , -2         /* move 2 months BACK */
                , 'B')       /* return DAY 1 of month */
```

To increase flexibility, how can I store a **date** value in a **MACRO** variable, so that it is available to my entire program?

```
/* macro variable with yyyymm for TWO months ago */
%let M2_yyyymm = %sysfunc(
                          intnx( MONTH
                                 , %sysfunc( today() )
                                 , -2
                               )
                         , yymmN6.   /*instruct %sysfunc how to format */
                         );
```

- Notice that a fourth INTNX argument is not needed, since the day of the month is not important
- When using INTNX in %SYSFUNC, do not use quotes for the arguments (such as MONTH)
- The second argument of %SYSFUNC controls the format of the result. YYmmN6 specifies a 4 digit year, 2 digit month, and NO separators.
- Result on March 10, 2018 is 201801 - 2 months prior to today

If a numeric date value is desired, rather than a format such as yyyymm, do not specify a format in the %SYSFUNC, and the result will be a SAS Date value.

```
/* macro variable with SAS Date Value - DAY 1 of month 2 months ago */
%let M2_yyyymm = %sysfunc(
                          intnx( MONTH
                                 , %sysfunc( today() )
                                 , -2
                                 , B
                                )
                         );
```

- Result on March 10, 2018 is 21185

CHANGING CHARACTER DATES TO SAS DATES

Frequently dates are provided as a character representation, instead of a numeric SAS Date representation. You can programmatically change the date to a SAS date using the **INPUT** function, together with the appropriate **informat** for reading the date.

```
data Change_Char_To_Date  (DROP = CharDate);
   CharDate = '1957-03-15';       /* date represented as yymmdd10. */
   putlog 'Char Value: ' chardate;
   /**convert to a SAS date **/
  SAS_date = input(chardate , yymmdd10.);
  Putlog  'Stored value of SAS_Date: '   SAS_date;
  putlog  'Format with ddmmyyS10.:  '    SAS_Date ddmmyyS10.;
run;
```

Results in Log:

Char Value: 1957-03-15

Stored value of SAS_Date:-1022

Format with ddmmyyS10.: 15/03/1957 ← *Note the S specifies SLASH as the separator*

DYNAMIC DATES IN FILENAMES

For easier file management, it is common to “date-stamp” your output files (logs, etc).

Step 1: Store the current date and time in a macro variable, using a format that displays as

Yyyymmdd_hhmm

Step 2: Use PROC PRINTTO to route your log to a permanent location with the customized file name.

```
%let DateTime =
       %sysfunc(
                compress(
                        %sysfunc(today(),yymmddN8.)_
                        %sysfunc(time(),hhmm6.)
                         , ': '          /* ← remove Colon and blank */
                        )
               );
   /* result looks like    20171115_1426   */
/* Note: N in yymmddN8 requests NO separators( Dash, Slash, etc)*/

** route Log to permanent location and version control**;
proc printto  log =
  "&Filepath_Out.\logs\&stage.\&Strategy_ID._&datetime..log"
;
run;
```

CHANGING HOW DATES ARE DISPLAYED

SAS date formats provide full control over how date values are displayed. The internal storage remains as an integer.

```
data dates;
   SystemDate = today();
   putlog 'Unformatted System Date is: ' SystemDate  ;
   putlog 'formatted with date9.     : ' SystemDate date9.    ;
   putlog 'formatted with worddate.  : ' SystemDate worddate. ;
   putlog 'formatted with weekdate.  : ' SystemDate weekdate. ;
   putlog 'formatted with weekdate9. : ' SystemDate weekdate9.;
run;
```

Log Results:

Unformatted System Date is: 21138

formatted with date9. : 15NOV2017

formatted with worddate. : November 15, 2017

formatted with weekdate. : Wednesday, November 15, 2017

formatted with weekdate9. : Wednesday

DETERMINE THE FISCAL YEAR OF A DATE

The SAS functions **Month** and **Year** are based upon CALENDAR years. Most businesses run on a FISCAL year which differs from the calendar year.

For example, for a Fiscal Year that begins in November, November 2017 would be considered **Month 1 of Fiscal Year 2018** and October 2018 would be considered **Month 12 of Fiscal Year 2018**.

To identify the Fiscal Year for a specified month, the following macro can be used.

```
/**** Create a Macro to compute fiscal year for a specified DATE
        START = beginning month for FY          ***/
%macro FY (date , start=7);                /** Macro with 2 parameters **/
   year(&date) + /* BELOW expression returns a 0 (False) or a 1 (True) */
        (month(&date) ge &start and &start ne 1)
%mend FY;
/*** example of macro usage ***/
data try_FY;
  Txn_date = '21nov2017'd;
  FiscalYear = %FY( Txn_Date , start = 11 ); /* macro call-like a function */
  putlog FiscalYear =  Txn_Date=date9.;
run;
```

FiscalYear=2018 Txn_date=21NOV2017

USING DATES TO CONTROL PROGRAM FLOW

We have looked at examples of computing dates, going forward and backward in time, and subsetting data based on dates, but one of the most powerful uses of dates is to control program flow. For example, suppose I have a daily process that runs 365 days a year. Every day it produces a Daily update (previous day's sales, etc), but on the first day of the month, we also want to produce a PRIOR month summary.

We could submit the Monthly summary on day one of every month , or we could include it in the daily job and control the process based upon the date at submission.

Example macro to control when to submit the monthly job:

```
*** Run monthly report on first of each month;
%macro dayone;
   %if %sysfunc(day( %sysfunc(today()) )) = 1
      %then %do;
         %include 'monthly_report.sas';  /*submit monthly job */
      %end;
%mend;
%dayone                          /* run DayOne macro */

*** Run daily report every day;
%include 'daily_report.sas';
```

CONCLUSION

Dates are an integral component of most programs, reports, and analytics. Understanding how SAS stores dates, processes dates, and enables date generalization makes our programming life easier.

This paper, along with the recommended reading, provides examples that you can hopefully integrate into your existing and new programs and processes.

ACKNOWLEDGMENTS

Thank you to Chris Hemedinger for his valued suggestions, based on my initial two presentations of this content. He included some great additions in his blog.

https://blogs.sas.com/content/sasdummy/2017/11/21/compute-date-past-future-intnx/

RECOMMENDED READING

- https://blogs.sas.com/content/sasdummy/2017/11/21/compute-date-past-future-intnx/
- https://communities.sas.com/t5/SAS-Nordic-Users-Group/Blog-article-Some-of-the-most-viewed-on-http-support-sas-com/gpm-p/400686
- https://web.archive.org/web/20080706004217/http:/support.sas.com/community/newsletters/news/insider/dates.html

CONTACT INFORMATION

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

Marje Fecht
Prowerk Consulting
Marje.Fecht@prowerk.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.