Paper 226-2007

Using SAS® Dates and Times – A Tutorial

Jonas V. Bilenas, JP Morgan Chase, Wilmington, DE

ABSTRACT

Using date and time constants and variables in SAS® is sometimes difficult to grasp even for the advanced SAS user. This tutorial will cover how to use date and time constants, INFORMATS, FORMATS, functions, using PICTURE format directives, how to set up date ranges in user formats, and how to use the %SYSFUNC macro function. We will also look at what is new in SAS®9 for handling dates and times.

INTRODUCTION

Use of dates and times in SAS can take the form of variable assignment from constant values or reading data from data lines or external files. These date and time variables can then be used in functions to calculate differences between 2 dates and or increment or decrease dates.

SAS date and time variables are stored in SAS as numeric data. Dates are expressed as integers and indicate the number of days since 1/1/1960. Time variables are represented as the number of seconds since midnight. Some variables contain both date and time elements and are referred to as datetime variables. Datetime variable indicate the number of seconds since midnight on 1/1/1960.

These numerical format of the data make it easy to do interval calculations but make for difficult interpretation when printed out. For that reason, we can use many date and time FORMATS to represent data in a format we are used to

Specifying Date and Time Constants

Let's see how to specify some date and time constants and assign these values to SAS variables.

```
options nocenter;

data _null_;
  car_svc_dt = '20DEC2006'd;
  car_scv_tm = '08:48't;
  car_scb_dt = '20DEC2006:08:48'dt;
  put _all_;
run;
```

The date specification is of the form 'DDMONYYYY'd or 'DDMONYY'd. Time specification takes on the form of 'hh:mm:ss't. Datetime variables need the dt extension in the statement of the constant. Look at the output:

```
car_svc_dt=17155 car_scv_tm=31680 car_scb_dt=1482223680 _ERROR_=0 _N_=1
```

Recall that the date represents days since 1/1/1960 and time represent seconds since midnight or midnight of 1/1/1960 if specifying a datetime variable. In the next section we can see how to use FORMATS to make the values more readable.

Using Date and Time FORMATS

If we are to represent the dates in a report, it would make more sense to report the dates and time in a format we are familiar with. Here are some SAS internal FORMATS we can use to display the data:

```
put   car_svc_dt= date9.
     /car_svc_dt= mmddyy9.
     /car_svc_dt= worddate18.
     /
     /car_scv_tm= hhmm5.
     /car_scv_tm= time8.
     /car_scv_tm= tod8.
     /
     /car_scb_dt= datetime16.
     ;
run;
```

The output listing is shown:

```
car_svc_dt=20DEC2006
car_svc_dt=12/20/06
car_svc_dt=December 20, 2006

car_scv_tm=8:48
car_scv_tm=8:48:00
car_scv_tm=08:48:00

car_scb_dt=20DEC06:08:48:00
```

Reading in Date and Time Variables Using Date and Time INFORMATS

If we use FORMATS to output data to reports of files then we need special INFORMATS to read in date and time variables. The following code illustrates just a few of the INFORMATS that SAS provides for reading in date and time data:

```
data _null_;
  input @1  car_svc_dt date7.
        @10 car_scv_tm time8.
        @20 car_scb_dt datetime16.
        ;
  put car_svc_dt= date9. +5 car_scv_tm= hhmm5. +5 car_scb_dt= datetime16.;
  datalines;
20DEC06  8:48      20DEC06:08:48:00
11oct06  9:34      20DEC06:09:11:14
;
run;
```

Output is as follows:

```
car_svc_dt=20DEC2006      car_scv_tm=8:48      car_scb_dt=20DEC06:08:48:00
car_svc_dt=11OCT2006      car_scv_tm=9:34      car_scb_dt=20DEC06:09:11:14
```

Sometimes our date data only provides month and year. There is an INFORMAT called MONYYw. that can be used to read in data that looks like AUG06. However, how would we go about reading data that has a numeric month instead of the character form (0806)? We can modify the variables and use an INPUT function to read the modified data:

```
proc format;
  value $mn '01' = 'JAN'  '02' = 'FEB'  '03' = 'MAR'  '04' = 'APR'
            '05' = 'MAY'  '06' = 'JUN'  '07' = 'JUL'  '08' = 'AUG'
            '09' = 'SEP'  '10' = 'OCT'  '11' = 'NOV'  '12' = 'DEC'
  ;
run;
data _null_;
  length stuff1 $3 stuff2 $2 stuff3 $5;
  input @1  stuffmmyy $char4.
        ;
  stuff1 = put(substr(stuffmmyy,1,2),$mn.);
  stuff2 = substr(stuffmmyy,3,2);
  stuff3 = stuff1 || stuff2;
  stuff  = input(stuff3,monyy5.);
  put stuffmmyy= stuff1= stuff2= stuff3= stuff=date9.;
  datalines;
1206
0905
;;;
run;
```

Results are as expected:

```
stuffmmyy=1206 stuff1=DEC stuff2=06 stuff3=DEC06 stuff=01DEC2006
stuffmmyy=0905 stuff1=SEP stuff2=05 stuff3=SEP05 stuff=01SEP2005
```

SAS9 introduced the ANYDTDTEw. INFORMAT to read in unusual dates. Here is an example:

```
data _null_;
  input dt anydtdte9.;
  put dt=date9.;
datalines;
03022007
02feb2007
0207
;
run;
```

Output is shown here. Note that the last entry was a replicate of the problem above where you get data in the form of mmyy but ANYDTDTE did not get desired results:

```
dt=02MAR2007
dt=02FEB2007
dt=.
```

YEARCUTOFF OPTION

We all remember the scares we had as Y2K was approaching near. We all had fears of cars not starting and coffee machines not brewing our favorite coffee. Old code that did not use SAS date and time variables explicitly had to changed. Years represented with 2 digits had to change to 4. For 2 digit years, the YEARCUTOFF option was modified to handle such dates.

The current default setting is YEARCUTOFF=1920. Any 2 digit year that is between 00 and 19 would have 20 as the first 2 digits of the 4 digit interpretation (i.e. 07 maps to 2007). Any 2 digit year that is from 20-99 will have the first 2 digits as 19 (i.e. 58 maps to 1958), Your best bet is to always use a 4 digit year.

Look at some previous code in the INFORMAT section if we were to change the YEARCUTOFF=1900. What happens with 20DEC06 and 11OCT06?

```
car_svc_dt=20DEC1906      car_scv_tm=8:48      car_scb_dt=20DEC06:08:48:00
car_svc_dt=11OCT1906      car_scv_tm=9:34      car_scb_dt=20DEC06:09:11:14
```

Date and Time Functions

We saw how to assign, read and output date and time variables, let's see how we can use these variables in date and time functions. Some of these functions can be used to generate SAS date and time variables from numeric or character data while other functions can calculate time and date intervals between 2 dates.

Creating Date and Time Variables from Numeric and/or Character Variables

Let's look at some code that takes character values for month day and year and combine all three to generate a SAS date:

```
data _null_;
  month='06';
  day='15';
  year='02';

  date = mdy(input(month,2.),input(day,2.),input(year,2.));
  put date=;
run;
```

Output from the source code:

```
date=15506
```

Similar functions work with time elements. This is illustrated with the following code:

```
data _null_;
  hrid=hms(12,45,10);
  put hrid
      / hrid time.
      ;
run;
```

Generated output:

```
45910
12:45:10
```

We can go the other way as well. We can take a date variable and extract individual components.

```
data _null_;
  dt = today();
  day = day(dt);
  month = month(dt);
  year = year(dt);
  qtr  = qtr(dt);
  weekday = weekday(dt);
  week = week(dt);
  put _all_;
run;
```

Note that the TODAY function returns the current date. The above source code generates the following output:

```
dt=17216 day=19 month=2 year=2007 qtr=1 weekday=2 week=7 _ERROR_=0 _N_=1
```

Date functions work on DATES and time functions work on TIME. If we apply a date function to a SAS DATETIME variable we will get errors. This is shown in the following log:

```
239  data _null_;
240    day = day('01feb94:8:45'dt);
241    put day=;
242  run;

NOTE: Invalid argument to function DAY at line 240 column 9.
day=.
day=. _ERROR_=1 _N_=1
```

To remedy the situation we can use the DATEPART function to extract only the date component.

```
245    day = day(datepart('01feb94:8:45'dt));
246    put day=;
247  run;

day=1
```

Interval Functions

There are a number of SAS functions that will enable one to calculate intervals between 2 dates or times. Let look at a few examples of these. The first looks at calculating an individuals age in terms of years and then in days.

```
data _null_;
  dob = '15dec1998'd;
  age = yrdif(dob,today(),'act/act');
  daysold = datdif(dob,today(),'act/act');
  put _all_;
```

```
run;
```

The output reports AGE and DAYSOLD as noted. Note that you will get a different result since I ran to above code In January of 2007.

```
dob=14228 age=8.1178082192 daysold=2965 _ERROR_=0 _N_=1
```

The details of the third argument of YRDIF and DATDIF specify how many days are allocated for the year and month. If we specify '30/360' we force each month to be counted with 30 days and each year to be set at 360 days. To use the actual days in each month and year, use the 'act/act' specification.

Another interval function is INTCK. This function is different that the DIF functions since it will report back the number of interval value boundary differences dependent on the first argument. Let's look at the results from the following code. Remember that the 2008 year is a leap year,

```
data _null_;
  lp  = '28feb2008'd;
  lp1 = '01mar2008'd;
  day_dif = intck('day',lp,lp1);
  mon_dif = intck('month',lp,lp1);
  yr_dif  = intck('year',lp,lp1);
  put _all_;
run;
```

Output:

```
lp=17590 lp1=17592 day_dif=2 mon_dif=1 yr_dif=0 _ERROR_=0 _N_=1
```

Note that the difference between the two dates is only 2 days but the reported month difference is 1 since the two dates jump one month boundary.

We can also increment a date, time or datetime variable and generate a new date, time or datetime variable based on the desired increment. This is accomplished with the INTNX function. Let's look at an example:

```
data _null_;
  lp  = '28feb2008'd;
  lpe =  intnx('month',lp,1);
  lpem = intnx('month',lp,1,'m');
  lpee = intnx('month',lp,1,'e');
  put lp=date9. lpe=date9. lpem=date9. lpee=date9.;
run;
```

Output is shown here:

```
lp=28FEB2008 lpe=01MAR2008 lpem=16MAR2008 lpee=31MAR2008
```

The third optional parameter to the INTNX function specifies the end point of the new variable; beginning (default), middle ('m'), or end ('e'). Each option is shown in the above output.

We can also use the INTNX function to subtract date and time values. As an application, let's prepare for Valentines Day. Here is some code to calculate the dates we have to prepare for. Valentines Day this year is '14FEB2007'. You are not even sure of the day of the week that falls on.

```
data _null_;
  dt = '14FEB2007'd;
  put dt=weekdatx29.;

dt=Wednesday, 14 February 2007
```

You will be in deep trouble if you are not prepared one day before that (in the same data step add):

```
/* subtract a day */
  back_1day = intnx('day',dt,-1);
  put back_1day=weekdatx29.;

back_1day=Tuesday, 13 February 2007
```

By that day, it is too late to plan anything. You figure that you need 8 workdays to start planning a dinner date. Calculate 8 days back from 2/14/2007 excluding weekends. The WEEKDAY argument skips over Saturday and Sunday. Code:

```
/* go back 8 workdays */
       back_8_workdays = intnx('weekday',dt,-8);
  put back_8_workdays=weekdatx29.;

back_8_workdays=Friday, 2 February 2007
```

You want to make a dinner date but you don't want to go out on Valentines, but the weekend before. Go back 1 week from DT and report on the end of the week. Alignment options for INTNX are beginning or b (default), middle or m, end or e, and sameday or s (added in SAS9).

```
/* go back one week and report last day */
  eopw = intnx('week',dt,-1,'e');
  put eopw=weekdatx29.;

eopw=Saturday, 10 February 2007
```

If Saturday is not available you can plan a back up reservation on Friday. You recall that, in SAS, dates are numbers representing days since 1/1/1960. What if we just subtract 1 from eopw:

```
eopwd = eopw-1;
put eopwd=weekdatx29.;

eopwd=Friday, 9 February 2007
```

Wow! What would we do without SAS to organize our lives?

Date and Time Directives in PICTURE FORMATS

You can add date and time information to your generated PICTURE FORMATS. Here is sample code and output:

```
proc format;
  picture dt
   low-high = 'TIME STAMP: %A %B %d, %Y.'
              (datatype=date)
  ;
  picture tm
    low-high = '%I:%M.%S%p'
               (datatype=time)
  ;

data _null_;
  file print;
  now = today();
  tm = time();
  put  now dt40.  tm tm.;
run;
```

Output:

```
TIME STAMP: Saturday January 27, 2007.12:39.42PM
```

From SAS Online Documentation for SAS9.1 we see there are a good number of directives we can add to our PICTURE FORMATS:

- %a Locale's abbreviated weekday name
- %A Locale's full weekday name
- %b Locale's abbreviated month name
- %B Locale's full month name
- %d Day of the month as a decimal number (1-31), with no leading zero
- %H Hour (24-hour clock) as a decimal number (0-23), with no leading zero
- %I Hour (12-hour clock) as a decimal number (1-12), with no leading zero
- %j Day of the year as a decimal number (1-366), with no leading zero
- %m Month as a decimal number (1-12), with no leading zero
- %M Minute as a decimal number (0-59), with no leading zero
- %p Locale's equivalent of either AM or PM
- %S Second as a decimal number (0-59), with no leading zero
- %U Week number of the year (Sunday as the first day of the week) as a decimal number (0,53), with no leading zero
- %w Weekday as a decimal number (1= Sunday, 7=Saturday)
- %y Year without century as a decimal number (0-99), with no leading zero
- %Y Year with century as a decimal number%%%

Related to user generated FORMATS, the following code shows how to specify date ranges in the value (or START – END) specification of user defined FORMATS and/or INFORMATS. This code sets up specific costs for specific dates within a data step:

```
proc format;
  value cost '01jan2007'd - '10jan2007'd = '1.10'
             '11jan2007'd - '20jan2007'd = '1.20'
             '21jan2007'd - '31jan2007'd = '1.30'
  ;

data _null_;
  cost = input(put(today(),cost.),4.2);
  put "&sysdate " cost=;
```

Output results:

```
27JAN07 cost=1.3
```

Using DATES and TIMES in Macros

You can also utilize DATES and TIMES in macros or in macro variables. One handy global macro variable to use is &sysdate to use system date in code. This gets declared automatically and can be used in open code as well. Here is an example where we add the system date to a SAS data set name.

```
data SGF_&sysdate;
  do i = 1 to 100;
    a = ranuni(1);
    output;
  end;
run;
```

LOG Output:

```
NOTE: The data set WORK.SGF_27JAN07 has 100 observations and 2 variables.
```

Within macro code you can also make use of %SYSFUNC function to assign date and time constants to macro variables. Here is an example:

```
proc contents data=work._all_;
title "Contents of data created today: %sysfunc(date(),worddate.)";
title2 "The time is: %sysfunc(time(),time.)";
run;
```

The generated title lines on the output:

```
Contents of data created today:   January 27, 2007
The time is: 13:50:19
```

You can also use the %sysfunc in a %let statement so that macro variables can also be used in open code as well. This example generates a title line for the run date. The second title specifies that the report is for the data through the end of the previous month.

```
%let month_end=%sysfunc(intnx(month,%sysfunc(today()),-1,e),yymmddd10.);
%let run_date=%sysfunc(date(),worddate.);

title1 Report Generated on &run_date;
title2 Report through &month_end;
```

OUTPUT:

```
Report Generated on February 19, 2007
Report through 2007-01-31
```

Some observations about the code listed for the example:

1. Arguments for an INTNX function within a %sysfunc function should not be quoted. The MONTH argument and the e (or end) should not be in quotes as is required in the data step.
2. The second argument of the first %sysfunc function is a format. This is handy since you don' t need a PUTN function to format the macro variable. You save some typing and parentheses.

CONCLUSION

Once you get a handle on how DATES and TIMES are stored in SAS variables you will be able to utilize the functionality of these variables in your code. There are many INFORMATS to read in date and time values and many FORMATS to output in readable form. There are a number of functions to use with date and time variables and you can use date and time directives in PICTURE FORMATS to gussy up reports. You can also work with global system macro variables and also generate new macro variable with %SYSFUNC and date and time functions.

REFERENCES AND ADDITIONAL INFORMATION:

- Bilenas, J. “The Power of PROC FORMAT”, SAS Press, 2005.
- Carpenter, A. “Looking for a Date? A Tutorial on Using SAS Dates and Times”, SUGI30, 2005.
- Morgan, D. “The Essential Guide to SAS Dates and Times“, SAS Press. 2006

CONTACT INFORMATION

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

Jonas V. Bilenas
JP Morgan Chase Bank
Wilmington, DE 19801
Email: Jonas.Bilenas@chase.com
jonas@jonasbilenas.com

This work is an independent effort and does not necessarily represent the practices followed at JP Morgan Chase Bank.