

BLIND DATES & OTHER LAPSES OF REASON: HANDLING DATES IN SAS

Alan Dickson, Atlantic Search Group, Inc.

Author's Note: This paper was originally presented on October 5th, 1992 at the Northeast SAS Users Group Conference in Baltimore, MD. Since the use of 10/5/92 as "today" is integral to several of the examples, no effort has been made to alter it.

INTRODUCTION

Although the title of this paper is obviously not entirely serious, it is not entirely irrelevant either. I hope to show that, when it comes to handling dates in SAS, a certain amount of blind faith is appropriate, if not essential. If you provide SAS with the information it needs, it will never let you down.

Also, SAS dates are the way they are for a reason - to facilitate the handling of lapses, i.e. the number of intervals between two dates. We'll look at some examples of simple, and not-so-simple, intervals. If you still feel uncomfortable with letting SAS handle your dates after that, you always have the option of writing your own algorithms in a DATA step - good luck.

THE BASICS

"A date is represented by the number of days between January 1, 1960 and that date." - page 85, SAS Language Reference, Version 6, First Edition. On first encountering this definition, nobody I've ever met said "Sorry, I can't mentally handle that concept." However, it's not uncommon for those same individuals to say weeks or months later "Oh - it really is handling them as numbers", or some similar exclamation.

Unfortunately, sometimes that lightbulb experience only happens after the customer complains about a report with sales figures reported as of 11,987.

Why did SAS choose to use 1/1/60 as the baseline for their date routines? I have no idea. However, I suspect that it was a conveniently recent date when the SAS system was first being developed, allowing for most contemporary dates to result in low positive integers, but with maximum future room for growth.

What happens before 1/1/60, you may ask? Simple - SAS stores the negative integer which results. The historical capacity is much less than the future one, but 1582 A.D. should be earlier than most requirements.

A THOUSAND WORDS WORTH

While not exactly a picture, some examples may help to demonstrate how this works (or doesn't). Note that only the GOODDATE variable is assigned both an informat and a format:

```
DATA TESTDATE;
INPUT GOODDATE BADDATE1 BADDATE2;
INFORMAT GOODDATE BADDATE1 MMDDYY6.;
FORMAT GOODDATE BADDATE2 MMDDYY8.;
CARDS4;
100592 100592 100592
;;;
PROC PRINT;
```

The output looks like this:

```
OBS GOODDATE BADDATE1 BADDATE2
1 10/05/92 11966 05/31/35
```

Clearly, GOODDATE is just the way we want it, BADDATE1 and BADDATE2 are not. BADDATE1 is showing exactly what SAS said it would, i.e. the number of days since 1/1/60. We could easily patch it up by applying the format to it in the PROC PRINT, or in another DATA step. But what about BADDATE2? It sure looks like a date but how did we get to a year of 35 instead of 92? This will shed some light:

```
PROC PRINT DATA=TESTDATE;
FORMAT GOODDATE BADDATE2 MMDDYY10.;
```

The output looks like this:

```
OBS GOODDATE BADDATE1 BADDATE2
1 10/05/1992 11966 05/31/2235
```

Now we can see that SAS took the value of 100592 and read it in as the number of days after 1/1/60 yet again, just as it said it would. Clearly, the moral here is - always give SAS the complete, correct information about how your dates are coming in, how you'd like to see them come out, and let SAS worry about how it handles them internally.

Another point to note here is that the First Edition manual indicates that the MMDDYY, DDMMYY & YYMMDD formats have a length limit of 8, but they will accept 10 (at least in the MVS implementation).

Try repeating the test code above with 65535 instead of 100592 as input for BADDATE2. The result you should get with the MMDDYY10. format is 06/06/2139. For those of you working in IBM MVS or CMS environments, the number 65535 should look suspiciously familiar. It is the largest integer that can be stored in a LENGTH 3 numeric variable.

Although it may seem counter-intuitive at first, you can take advantage of this fact to store your dates in only 3 bytes each (assuming that your code will be replaced sometime in the next 150 years or so).

```
DATA TESTDAT2;
LENGTH TODAY FUTURE 3;
INPUT TODAY FUTURE;
INFORMAT TODAY MMDDYY6.;
FORMAT TODAY FUTURE MMDDYY10.;
CARDS4;
100592 65535
;;;
PROC PRINT;

OBS TODAY FUTURE
1 10/05/1992 06/06/2139
```

DATE CONSTANTS

So much for reading and writing dates with SAS - let's start manipulating them. As mentioned earlier, one of the main reasons for the way SAS handles dates is to make lapse calculations easy - you simply subtract one date from the other.

Comparison of dates is equally easy. Later dates are stored internally as increasingly larger integers. Thus the regular arithmetic comparison operators are all you need (< > = EQ NE LT LE GT GE).

But what if one of the dates is not an input - for instance, a fixed date such as start of business year (or month), or even today's date? Again, SAS makes it easy.

To specify a fixed date, use the form "ddMONyy"D, e.g. "21FEB54"D. Although the quotes can be single or double, the format is rigid - "210254"D will result in an error message. Failing to include the trailing D identifier will result in SAS trying to convert the quoted value to numeric and then treating it as a number of days.

To specify the current date, SAS provides two identical functions, TODAY() and DATE(). Here's an example:

```
DATA _NULL_;
TODAY1 = TODAY();
TODAY2 = '05OCT92'D;
MY_DOB = '21FEB54'D;
DAYS_OLD = TODAY1 - MY_DOB;
PUT TODAY1= MMDDYY8.;
PUT TODAY2= MMDDYY8.;
PUT DAYS_OLD=;
giving
TODAY1 = 10/05/92
TODAY2 = 10/05/92
DAYS_OLD = 14106
```

THE SUM OF THE PARTS

In addition to providing a wealth of formats and informats for dates, SAS has a wide selection of functions for splitting out the component parts of a date, putting a date together from such parts, and translating dates from one form to another.

Continuing on with our previous example:-

```
MY_B_YR = YEAR(MY_DOB);
MY_B_QTR = QTR(MY_DOB);
MY_B_MTH = MONTH(MY_DOB);
MY_B_DAY = DAY(MY_DOB);
MY_B_WDAY = WEEKDAY(MY_DOB);
MY_BIRTH = MDY(MY_B_MTH,
MY_B_DAY,MY_B_YR);
PUT MY_B_YR=;
PUT MY_B_QTR=;
PUT MY_B_MTH=;
PUT MY_B_DAY=;
PUT MY_B_WDAY=;
PUT MY_BIRTH= MMDDYY8.;
PUT MY_BIRTH= WORDDATE18.;
```

This results in:

```
MY_B_YR = 1954
MY_B_QTR = 1
MY_B_MTH = 2
MY_B_DAY = 21
MY_B_WDAY = 1 (i.e. Sunday)
MY_BIRTH = 02/21/54
MY_BIRTH = February 21, 1954
```

Some systems (e.g. tape management systems) care little for such human niceties as weekends or holidays and use the julian date system instead of the normal "gregorian" system. For anyone who has never encountered this system before, it represents dates as 2 or 4 year digits followed by the number of days into the year (1-366) e.g. 92279. SAS provides the JULDATE and DATEJUL functions to translate between them:

```
MY_B_JUL = JULDATE(MY_DOB);
JULIAN = DATEJUL(92365);
PUT MY_B_JUL=;
PUT JULIAN= MMDDYY8.;
giving
MY_B_JUL = 54052
JULIAN = 12/30/92
```

Note that JULIAN does not return December 31st, since 1992 is a leap year. In fact, the whole problem of dealing with leap years - and soon, leap century - is one very good reason for leaving it all up to SAS. Does anyone still feel like trying to do this in a DATA step?

INTERVAL TRAINING

We have seen how easy it is to determine how many days there are between two dates in SAS - you simply subtract one from the other. Similarly, if you want to derive a date which is a certain number of days in the future, you just add the number to today's date, or the date you are using as a baseline.

This works fine for such applications as calculating:

21-day due date on library books,
30-day grace period on credit card payments,
60-day lapse dates on insurance coverage,
90-day warranty periods, etc.

However, there are many other intervals in use in different businesses. SAS has supported the most common interval units for some time, and SAS 6.07 has added several new ones. I recommend you examine the P-222 Technical Report-Changes and Enhancements to Base SAS Software manual for all the details.

SAS 6.07 has also added the flexibility of multi-unit intervals and shifted intervals. These options can also be combined to generate a shifted, multi-unit interval. These can be difficult to appreciate at first, so let's try to work up to them slowly.

SAS provides two functions for dealing with intervals, INTCK and INTNX. Quite what they stand for is never really spelled out in the manuals, but I think of them as INTerval ChecK and INTerval NeXt (although it could also be iNdeX). INTCK tells you how many intervals there are between two dates, INTNX tells you the date that results by moving one date a certain number of intervals, either forward or backward in time.

It is important to realize that the rules which SAS uses to define intervals are based on how many interval boundaries lie between dates, not on absolute numbers of days. For instance, between the NESUG '92 conference and my birthday in 1993, one year boundary is crossed, so

```
X = INTCK('YEAR', '21FEB92D', '05OCT92D');
Y = INTCK('YEAR', '05OCT92D', '21FEB93D');
```

will return a value of 0 in X and 1 in Y, even though X represents almost twice as long a period of time as Y.

The boundaries are generally the first day of each period, with Sunday being the first day in a week. This also applies to the effect of the INTNX function. Its use will advance the from-date through the specified number of interval boundaries and return the date of the first day of the next interval. Again, this can lead to some unexpected results if not borne in mind.

For instance,

```
Z1 = INTNX('MONTH', '29FEB92D', 1);
Z2 = INTNX('MONTH', '29FEB92D', -1);
```

will give Z1 the value corresponding to March 1st, 1992 rather than March 29th. Z2 will equate to January 1st.

I'm not suggesting here that there is anything wrong with these rules. Obviously, some system had to be chosen and this one is probably the best option. It can cause some confusing results though, unless you are careful.

MULTI-UNIT and SHIFTED INTERVALS

If you need to advance a date by more than one interval, the simplest approach is to use a different number as the last parameter in the INTNX function statement, e.g.:

```
Z = INTNX('MONTH', '29FEB92D', 6)
```

would take you to August 1st. That last parameter can also be a variable however, and there may be situations where a multi-unit interval would make the program easier to build and maintain. For instance, to do three-year followups of patients, a doctor might want to build a file using multi-unit intervals. These are formed simply by adding the multiplier number to the interval specification. E.g.

```
DATA PATIENT.FOLLOWUP;
  ARRAY CHKUP {10} CHKUP1-CHKUP10;
  START = '01JAN93D';
  DO I = 1 TO 10;
    CHKUP{I} = INTNX('YEAR3', START, I);
  END;
```

Multiplying I by 3 and using the YEAR interval would give the same result, but this code is slightly easier to follow. Note that this is still based on 1/1/60. If START was not a three-year point, the interval would need to be shifted to align the results correctly.

Shifted intervals allow you to specify that an interval is to start somewhere other than the first. They are formed by adding a period and a shift number after the interval specification e.g. YEAR.3 would be useful for a business whose fiscal year ran from March 1 - February 28 (or 29).

Combining these two forms provides shifted multi-unit intervals. The P-222 manual mentions using YEAR4.11 to create presidential election periods - a good example:

```
N = INTCK('YEAR4.11', MY_DOB, '10OCT92D');
```

indicates that there have been 9 elections in my lifetime.

Remember, in calculating multi-unit intervals, SAS still bases the results on the base date of 1/1/60. Thus,

```
P = INTCK('YEAR4.11',MY_DOB,'25DEC92D');
```

will give P the value 10, although I have not been around for 10 4-year periods. The reason for this is that I didn't have to wait a full 4 years for the first election. Similarly,

```
Q1 = INTCK('MONTH2','05OCT92D','05JAN93D');
Q2 = INTCK('MONTH2.2','05OCT92D','05JAN93D');
```

will yield 2 for Q1, but only 1 for Q2. The Q1 check crosses boundaries at 11/1 and 1/1. The Q2 check only crosses one at 12/1.

MY ONLY WEEKNESS

Although 1/1/60 was the first of a month, quarter, semi-year, year and even an election year, it was not a Sunday. This causes a slight wrinkle in the handling of multi-week intervals. SAS has chosen to use the Sunday of the week containing 1/1/60 in this situation i.e. December 27, 1959.

If that causes your multi-week intervals to go askew, you can always shift them back into line. However, since weeks align themselves differently within the year each year, it is an awkward interval under any circumstances. I refer you to the Q&A section (p.48) of the First Quarter '92 edition of SAS Communications for more on this topic - such as why SAS has no WEEK function.

THE WAIVE OF THE FUTURE

Let's look now at an example from the real world. This is based on some work I did in the insurance industry. When presented with the results, the user agreed that it was what they asked for but, sadly, it was not what they wanted. They finished up doing something entirely different but, rather than let the code go to waste, I present it to you.

While my insurance knowledge is less than stunning, let me attempt to explain a premium waiver to those of you in other industries. If you are in insurance, it may be best if you don't pay too much attention to this section.

Basically, the premise is that if you get disabled, it would be unfair to pay your claim on one hand, while taking some of it back to keep your insurance in-force on the other. Thus, many policies have a premium waiver clause which relieves you from contributing while you are on claim.

In this example, the user wanted to derive a date five months from the date of incurral of the injury or illness. On this date, the claim was to be reviewed to ensure that the appropriate waiver action was being taken.

The definition of five months was just that - not 150 days or 21 weeks, but the same day of the month, 5 months on.

This obviously begs the question - What if there is no corresponding day of the month in the fifth month? The user decided that the day should be backed down to the 30th for 30-day months and the 28th for February, even if it was a leap year.

Here is the essence of the code I developed:

```
DATA PREMWAIV (KEEP=INCURRAL PW_DATE);
FORMAT INCURRAL PW_DATE MMDDYY8.;
INPUT INCURRAL MMDDYY6.;
I_DAYS = DAY(INCURRAL);          /* Get day */
I_MONTH = MONTH(INCURRAL);       /* Get mth.*/
IF (I_DAYS > 28
AND I_MONTH = 9)
THEN I_DAYS = 28;                /* Fix Feb.*/
ELSE
IF (I_DAYS > 30
AND (I_MONTH = 1
OR I_MONTH = 4
OR I_MONTH = 6
OR I_MONTH = 11))
THEN I_DAYS = 30;                /* Fix rest */
/* Move to 1st of 5th month */
PW_DATE = INTNX('MONTH',INCURRAL,5);
/* Move to adjusted day in it */
PW_DATE = INTNX('DAY',PW_DATE,I_DAYS - 1);
CARDS4;
013192
022192
093092
100592
;;;
```

PROC PRINT;

The results are:

OBS	INCURRAL	PW_DATE
1	01/31/92	06/30/92
2	02/21/92	07/21/92
3	09/30/92	02/28/93
4	10/05/92	03/05/93

Although it is not immediately apparent, only the incurral months of January and September actually need adjusting, since each of the other situations involve a 30-day month at both points, i.e. a potential problem date of November 31st could only be generated by a date of June 31st.

That should never happen if the input dates are valid. The same situation holds for November-April and April-September. I left the adjustment code in for completeness and clarity.

DATING & CARS - AN AMERICAN TRADITION

The premise of my final example is this - a joint venture of auto makers has led to a new Japanese car assembled and finished in Taiwan. They are now test-marketing it in Europe and America and everyone involved is of course anxious to see how it's selling and what the projections are. (The source code is shown in Fig.1 on the following page. The final output is on Fig. 2, following Fig. 1).

The sales figures have already been consolidated for the two test markets and the data exists in a file called AUTO. It consists of a character flag indicating the continent, the sale date and the number of cars sold. It looks something like this:

```
A 090192 25
E 010992 21
A 090292 22
E 020992 20
A 090392 26
E 030992 22
A 090492 27
E 040992 21
etc.....
```

Notice that the dates on the European figures are in European form i.e. Day Month Year and have to be read with the DDMMYY. format. The rest of the processing is as follows.

Read the data and create the end of the 10-day period in which auto dealers report sales using the new TENDAY interval in SAS 6.07. Summarize the sales by 10-day periods. Adjust the sales to standardize on 9 sales days in each 10-day period, since most dealers are open Saturday but closed Sunday. This will scale up those periods in which 2 Sundays fell. Later, we will scale back down the projections in those periods which have only 8 sale days. Calculate the average increase in sales over time.

The calculation of the scaling factor is the only tricky piece of code here - let's look at it in a bit more detail to see how it is handled. We use the INTCK function to count the number of sale days with the new SAS 6.07 interval WEEKDAYnW, which allows you to define non-standard weekend days by substituting the relevant numbers for n. In this case, we use WEEKDAY1W to count 6-day work weeks with Sunday off.

We generate the from-date as the start of the current period by using the INTNX function with an offset of zero. Remember, it always results in a period start. Since we already have the end of period, all we have to do is add 1 to it to generate the to-date. We could also have used the INTNX function with an offset of 1 to achieve the same result.

After the real data is used up, we'll generate projections for the next 4 months, scaling as appropriate. Note that the last week of December actually scales up because there are 10 weekdays between the 21st and the 1st.

Obviously, this simple projection algorithm doesn't cater for holidays.

Finally, we print out the actual and projected figures, with the as-of dates printed multiple times with the appropriate format for each of the nationalities involved. NENGO is used in Japan and represents the year, month and day of an emperor's reign (in this case, Heisei). MINGUO is a Taiwanese date format based on 1910.

THE KISSOFF - YEARCUTOFF

This option showed up in SAS 6.07 with no fanfare, but it could just save a lot of people many headaches over the next ten years. It simply defines where the 100-year window should break when only 2-digit years are read. It is easier to show it in action than to explain:

```
OPTIONS YEARCUTOFF=1950;
DATA _NULL_;
INPUT CENTDATE MMDDYY6.;
FORMAT CENTDATE MMDDYY10.;
CARDS4;
100545
100554
100592
100505
;;;
PROC PRINT;
giving
OBS  CENTDATE
1    10/05/2045
2    10/05/1954
3    10/05/1992
4    10/05/2005
```

With the average lifespan increasing, this will not solve all problems in industries such as insurance and health care, but in many other areas, this simple option may save you a lot of code changes and file rebuilding. Enjoy.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the support of L.L. Bean Inc., of Freeport, Maine for graciously allowing me the use of their facilities and resources in the preparation of this paper.

SAS is a registered trademark of SAS Institute, Cary, NC.

Fig. 1

```

DATA AUTODATA;
  INFILE AUTO;
  INPUT @1 CONT $1. @;
  IF CONT = 'A'
  THEN INPUT @3 SALEDATE MMDDYY6.
           @10 CARSSOLD 6.;
  ELSE INPUT @3 SALEDATE DDMMYY6.
           @10 CARSSOLD 6.;
  ASOFDATE = INTNX('TENDAY',SALEDATE,1)-1;          /* GENERATE END OF SALES PERIOD */

PROC SUMMARY NWAY;
CLASS ASOFDATE;
VAR CARSSOLD;
OUTPUT OUT=TENDAYS SUM=;                          /* SUMMARIZE BY SALES PERIODS */

DATA ACT_PROJ (KEEP=ASOFDATE CARSSOLD BASIS SCALEUP SCALEDN);
SET TENDAYS END=END_FILE;
SCALEUP = 9/INTCK('WEEKDAY1W',
  INTNX('TENDAY',ASOFDATE,0),ASOFDATE+1);          /* SEE TEXT */
CARSSOLD = ROUND(CARSSOLD*SCALEUP);                /* GENERATE STANDARDIZED NOS. */
LASTSALE = LAG(CARSSOLD);                          /* CALCULATE INCREASES */
IF LASTSALE NE .                                  /* FROM ONE PERIOD */
THEN DO;                                           /* TO THE NEXT */
  TIMESINC+1;
  SALESINC+((CARSSOLD-LASTSALE)/LASTSALE);
END;
BASIS = 'ACTUAL ';
OUTPUT;                                           /* NO FURTHER PROCESSING */
IF NOT END_FILE                                /* FOR REAL DATA */
THEN RETURN;
AVRG_INC = SALESINC / TIMESINC;                  /* CALCULATE AVG. INCREASE */
BASIS = 'PROJECTED';
DO I = 1 TO 12;
  ASOFDATE = INTNX('TENDAY',ASOFDATE,2)-1;          /* PROJECT DATES */
  SCALEDN = INTCK('WEEKDAY1W',
    INTNX('TENDAY',ASOFDATE,0),ASOFDATE+1)/9;      /* SIMILAR TO SCALEUP */
  CARSSOLD+ROUND(CARSSOLD*AVRG_INC*SCALEDN);        /* STANDARDIZE PROJECTIONS */
  OUTPUT;
END;

PROC PRINT;
FORMAT ASOFDATE MMDDYY8.;                      /* PRINT RAW DATA */

DATA _NULL_;
SET ACT_PROJ;
FILE PRINT;
IF _N_ = 1                                     /* PRINT FORMATTED DATA */
THEN PUT @21 'ACTUAL & PROJECTED CARS SOLD'
      / @18 'COMBINED AMERICAN & EUROPEAN SALES'
      // @26 '----- AS OF DATE -----'
      / @01 'BASIS CARS SOLD (AMERICA)'
      @38 '(EUROPE) (JAPAN) (TAIWAN)' //;
PUT @01 BASIS @13 CARSSOLD COMMA7. @27 ASOFDATE MMDDYY8.
  @38 ASOFDATE DDMMYY8. @50 ASOFDATE NENG010. @62 ASOFDATE MINGUO8.;

```

Fig. 2

OBS	ASOFDATE	CARSSOLD	BASIS	SCALEUP	SCALEDN
1	09/10/92	436	ACTUAL	1.000	.
2	09/20/92	504	ACTUAL	1.125	.
3	09/30/92	527	ACTUAL	1.000	.
4	10/10/92	574	PROJECTED	1.000	0.88889
5	10/20/92	632	PROJECTED	1.000	1.00000
6	10/31/92	696	PROJECTED	1.000	1.00000
7	11/10/92	766	PROJECTED	1.000	1.00000
8	11/20/92	843	PROJECTED	1.000	1.00000
9	11/30/92	919	PROJECTED	1.000	0.88889
10	12/10/92	1012	PROJECTED	1.000	1.00000
11	12/20/92	1103	PROJECTED	1.000	0.88889
12	12/31/92	1227	PROJECTED	1.000	1.11111
13	01/10/93	1337	PROJECTED	1.000	0.88889
14	01/20/93	1472	PROJECTED	1.000	1.00000
15	01/31/93	1620	PROJECTED	1.000	1.00000

**ACTUAL & PROJECTED CARS SOLD
COMBINED AMERICAN & EUROPEAN SALES**

BASIS	CARS SOLD	----- AS OF ----- (AMERICA) (EUROPE)	DATE ----- (JAPAN) (TAIWAN)
ACTUAL	436	09/10/92	10/09/92
ACTUAL	504	09/20/92	20/09/92
ACTUAL	527	09/30/92	30/09/92
PROJECTED	574	10/10/92	10/10/92
PROJECTED	632	10/20/92	20/10/92
PROJECTED	696	10/31/92	31/10/92
PROJECTED	766	11/10/92	10/11/92
PROJECTED	843	11/20/92	20/11/92
PROJECTED	919	11/30/92	30/11/92
PROJECTED	1,012	12/10/92	10/12/92
PROJECTED	1,103	12/20/92	20/12/92
PROJECTED	1,227	12/31/92	31/12/92
PROJECTED	1,337	01/10/93	10/01/93
PROJECTED	1,472	01/20/93	20/01/93
PROJECTED	1,620	01/31/93	31/01/93