BASIC AND ADVANCED USES OF SASDATES

Kenneth L. Goodwin, Inland Steel Company

Introduction

In my current position, I am frequently asked to retrieve and report on information by date, whether it is to give summaries by ship week, report production for a given week, or produce charts for annual reporting. I have found numerous methods for employing SASdates to accomplish these tasks, including using them to reduce or eliminate the need for date-related program maintenance. This paper will explain SASdates and their basic uses, plus illustrate some more advanced ways to utilize them to accomplish common tasks more efficiently.

The Basics of SASdates

SASdate Structure

The SAS[1] System can read, write, and work with values that represent dates. These "SASdates" are values based on the number of days before or since January 1, 1960. Figure 1 shows the corresponding SASdates for some sample dates.

Calendar Date	**SASdate value**
January 1, 1959	-365
January 1, 1960	0
January 1, 1961	366
March 18, 1997	13591

Figure-1

The use of a sequence number to represent a date is not unique to SAS. Other software packages such as FOCUS[2] and LOTUS 1-2-3[3] also utilize this concept. The starting date used is different but the usage is very similar. When I first learned of this representation for dates it seemed unusual, but as I began to use SASdates their power became very apparent. They enable you to make date comparisons very easily without having to worry about year turnover, leap year adjustments, new centuries, etc. Through the use of the many date functions, you can use all or part of the date as needed, and you can utilize the various date formats to ease date output. Additionally, you can subgroup dates with formulas instead of employing a long series of IF statements.

Getting the Date In

The first step is to transform the date from the format in which it is stored to a SASdate value. This can be done in many ways, three of which are explained here.

The first method is to use one of the date INFORMATs in an INPUT statement, reading in the dates from input records. Figure-2 shows some sample date INFORMATs and their associated data forms.

```
DATA;

INPUT  @4    DATEA_1   DATE7.
       @17   DATEA_2   DATE11.
       @35   DATEA_3   DATE9.  /
       @4    DATEB_1   DDMMYY6.
       @17   DATEB_2   DDMMYY10.
       @35   DATEB_3   DDMMYY8.  /
       @4    DATEC_1   MMDDYY6.
       @17   DATEC_2   MMDDYY10.
       @35   DATEC_3   MMDDYY8.  /
       @4    DATED_1   YYMMDD6.
       @17   DATED_2   YYMMDD10.
       @35   DATED_3   YYMMDD10.;

CARDS;
   13SEP93    13 SEP 1993     13-SEP-93
   130993     13 09 1993  12 09 93
   091393     09 13  1993 09-13-93
   930913     1993 09 13  1993/09/13
   ;
```

Figure-2

Date functions are another method of creating SASdates.

Function	
DATE(), TODAY()	Returns the current system date as a SASdate value.
DATEJUL(yyddd)	Returns the SASdate value from a Julian date value.
MDY(mm,dd,yy) 'ddmmmyy'D	Returns the SASdate value for the entered month, day and year values.

Lastly, if you need to bring in a date that is stored in a packed, or binary, field you can use either of the approaches shown below to create a SASdate. You will notice, however, how much more straightforward and streamlined the second option would be. (The input date "datein" is in the form of yyyymmdd.)

```
1)      YEAR=INT(datein/10000);
        MONTH=INT(datein/100)-YEAR*100;
        DAY=datein-YEAR*10000-MONTH*100;
        DATE=MDY(MONTH,DAY,YEAR);

2)      DATE=INPUT(PUT(datein,8.),YYMMDD8.);
```

This latter example saves three statements, the creation of three variables, and utilizes the powerful PUT() function, which I use frequently.

Getting the Date Out

Just as there are INFORMATs which are used to input dates, there are similar FORMATs to output dates. Some of the common FORMATs are shown in Figure-3, along with examples of their output form.

Format	**Examples**
DDMMYYw.	180397, 18/03/1997
MMDDYYw.	03/18/97
MONYYw.	MAR97, MAR1997
WEEKDATEw.	Tuesday, March 18, 1997
WORDDATEw.	Mar 18, 97
YYMMDDw.	970318
YYQw.	97Q1

Figure-3

These formats can take on extended forms, depending on the width value (w.) used. This can be illustrated using the WEEKDATE format.

Format	**Value Printed**
WEEKDATE3.	Tue
WEEKDATE9.	Tuesday
WEEKDATE15.	Tue, Mar 18, 97
WEEKDATE17.	Tue, Mar 18, 1997
WEEKDATE23.	Tuesday, Mar 18, 1997
WEEKDATE29.	Tuesday, March 18, 1997

Date FORMATs must be used to print out SASdates. Without these FORMATs, SASdates would merely print out as the sequence number representing the number of days before or since January 1, 1960. Figure-4 illustrates what happens when you neglect to use a date FORMAT to print dates.

```
DATA SERVICE;
  INPUT       @1    NAME        $10.
              @15   HIREDT      MMDDYY8.
              @25   LASTDT      YYMMDD6.
  SERV = LASTDT - HIREDT;
  YRS = SERV / 365.25;

CARDS;
CADE, J       12/01/73    851231
RAY, M         8/3/77 860102
;

PROC PRINT;

OBS  NAME     HIREDT      LASTDT    SERV    YRS
 1   CADE, J  5083        9496      4413    12.08
 2   RAY, M   6424        9498      3074     8.42
```

Figure-4

Notice that the amount of time served in days and years is printed just fine, but the HIREDT and LASTDT values are not very useful printed as SASdate values. This example illustrates two points: there is a difference between "dates" (e.g. HIREDT, LASTDT) and "duration" (SERV, YRS), and printing a SASdate in a understandable form requires the use of a date FORMAT.

To correct the date formatting issue in this example, add "FORMAT HIREDT LASTDT DATE.;" to the PROC PRINT statement, as shown in Figure-5. The results are much more useful.

```
DATA SERVICE;
  INPUT       @1    NAME        $10.
              @15   HIREDT      MMDDYY8.
              @25   LASTDT      YYMMDD6.
  SERV = LASTDT - HIREDT;
  YRS = SERV / 365.25;

CARDS;
CADE, J       12/01/73    851231
RAY, M         8/3/77 860102
;

PROC PRINT;
 FORMAT HIREDT LASTDT DATE.;


OBS  NAME     HIREDT      LASTDT    SERV     YRS
 1   CADE, J  01DEC73     31DEC85   4413     12.08
 2   RAY, M   03AUG77     02JAN86   3074      8.42
```

Figure-5

Creating Record Selection Criteria

The ability to use SASdates in your record selection statements adds another powerful tool to your tool kit. SASdates can be part of a simple statement such as

```
IF BORN_DT LT '01JAN89'D ;
```

Or the statements can be more involved like this example, which is used to automatically select all records from the previous month.

```
TODAY=TODAY() ;
* last day of previous month ;
END  =TODAY-DAY(TODAY) ;
* first day of previous month ;
START=END-DAY(END)+1;
IF datein GE START AND datein LE END ;
```

These are just two examples of selection criteria. I'm sure that you can develop many more.

SASdate Functions

As you can see I have used a few of the many functions that are available for manipulation of a SASdate. To go into detail on all of them is beyond the scope of this paper, but I will list several and recommend that you review your documentation for more detail covering these and other SAS date functions.

DAY	Returns the day of the month from a SASdate value.
INTCK	Returns the number of specific time intervals between two SASdates (e.g. weeks, months, etc.).
INTNX	Returns the SASdate value a given number of specific time intervals from a starting date (e.g. SASdate of 3 weeks ago).
MONTH	Returns the month from a SASdate value.
QTR	Returns a 1,2,3 or 4, depending on the quarter, from a SASdate value
WEEKDAY	Returns the day of the week from a SASdate value

Advanced Uses for SASdates

Using SASdates to Replace IF Statements

Subsetting information for processing based on date is often done using IF statements. Using SASdates can make the subsetting easier, more concise, and removes the need for updating the IF statements as time passes.

1) Subgrouping by week

Since I am frequently asked to subgroup data based on week, I looked for an efficient method of subgrouping by week without using a complex set of IF statements. As a result, I developed an algorithm to determine the week, using Sunday as the first day of the week.

In essence, this algorithm assigns the same SASdate (Sunday's) to the WEEKOF variable for each day of the week. In that way, you can quickly use the WEEKOF variable to subgroup by week.

```
WEEKOF = INT((datein-2)/7)*7+2;
```

While this looks awkward, it can be best explained by thinking of it as a way to assign the same integer value to seven different days. Hence, the "INT(x/7)*7" portion of the algorithm. What is initially confusing is the introduction of an offset of 2 into the algorithm. SASdate zero, or January 1, 1960, was a Friday. Consequently, the first Sunday was January 3rd, and so all must be offset two days to align with a Sunday. Therefore, the algorithm first subtracts two days from the "datein", performs the "INT(x/7)*7" function to assign the same integer to each day of the week, and them adds back the two offset days originally subtracted. The result is that WEEKOF contains the Sunday SASdate for each day of that week.

You can get the same result using the INTNX function.

```
WEEKOF = INTNX('WEEK',datein,0) ;
```

The INTNX function is used to advance a date by a given interval and takes the form

```
INTNX(interval,start date,number of intervals).
```

The interval can any of the following: DAY, WEEK, WEEKDAY, MONTH, SEMIMONTH, QTR, SEMIYEAR, or YEAR. The start date has to be a valid SASdate value, and the number of intervals is a positive or negative value that represents the specific number of time intervals to move from the start date. In this case, moving 0 time intervals results in the function returning the SASdate for the Sunday of that week.

Figure 6 shown on the next page shows the results of this exercise.

DATEIN	**SASdate**	**WEEKOF**
08MAR92	11755	11755
14MAR92	11761	11755
28DEC86	9858	9858
03JAN87	9864	9858

Figure-6

2) Creating Month Variables

If you just need to know the month of a given date, the MONTH function works directly with the SASdate and returns a numeric value representing the month of the year (e.g. 1 for January, 2 for February, etc.).

```
MONTHDAT = MONTH(datein);
```

If, however, you need the month to be displayed in a 3-letter character format, you can use one of two methods:

```
IF MONTH = 1 THEN CHARMON = 'JAN';
IF MONTH = 2 THEN CHARMON = 'FEB';
IF MONTH = 3 THEN CHARMON = 'MAR';
                 .....
IF MONTH = 12 THEN CHARMON = 'DEC';
```

This is the traditional method of determining the appropriate month name. It is straightforward, but it is lengthy.

```
CHARMON = PUT(datein,WORDDATE3.);
```

This method is much shorter, and puts to use the power of the PUT() statement and the WORDDATE INFORMAT.

<u>Creating Alternative Date Representations</u>

As described so far, SASdates, date formats, and output statements enable you to print dates in many varied forms. Sometimes, however, the form you want is not available as a SASdate format. In these instances, you can develop brief algorithms to manipulate the SASdates and create a new variable which contains the date in the desired format. Shown below are several examples of how to accomplish this. Note the importance of clear code and liberal comments.

1) YY/MM form

The traditional method of creating a variable containing a date in the YY/MM form would be to use a series of IF statements.

```
IF DATE GE 930101 AND DATE LE 930131
     THEN CHARMON = '93/01';
IF DATE GE 930201 AND DATE LE 930228
     THEN CHARMON = '93/02';
      .........
IF DATE GE 931201 AND DATE LE 931231
     THEN CHARMON = '93/12';
```

While effective, this code becomes a maintenance nightmare, requiring an update each year to put in the new year's value. It also requires a lot of statements to derive the correct value. Contrast this with the following statements that utilize SASdate functions.

```
YEAR = PUT(datein,YYMMDD2.);  *returns yy;
MON  = PUT(datein,MMDDYY2.);  *returns mm;
CHARMON = YEAR||'/'||MON;  *forms yy/mm;
DROP YEAR MON;
```

This code is concise, straightforward, and requires no date-related maintenance. It also includes the lesser-known use of the YYMMDD2. and MMDDYY2. formats to return just the yy and mm, respectively.

2) MMM_YY form

A series of IF statements can also be used to create the MMM_YY form.

```
IF DATE GE 930101 AND DATE LE 930131
     THEN CHARMON = 'JAN 93';
IF DATE GE 930201 AND DATE LE 930228
     THEN CHARMON = 'FEB 93';
     ...........
IF DATE GE 931201 AND DATE LE 931231
     THEN CHARMON = 'DEC 93';
```

As mentioned above, this code will require annual maintenance to update it for the new year, and it requires many statements. The following SASdate algorithm accomplishes the same date manipulation in only four statements, and again requires no date-related maintenance.

```
  * returns 3-char month with space;
MON  = INPUT(PUT(datein,WORDDATE3.),$4.);
  * returns yy;
YEAR = PUT(datein,YYMMDD2.);
  * forms mmm_yy;
CHARMON = MON||YEAR;
DROP  MON YEAR;
```

Note that the first line of the algorithm combines both the PUT() and INPUT() functions to return the three-character month name using a four-character INFORMAT, thereby adding a space to the end.

3) NN and YYYYNN form

I do a significant amount of reporting subgrouped by calendar week number. The method that is used to determine the numeric numbering of calendar weeks is tricky at best, and confusing at worst. In trying to come up with an algorithm to calculate week number (nn) and week number with year (yyyynn), I must admit that I went through many attempts before arriving at the one listed in Figure-7. And while most of the code is fairly straightforward, the parts of it that handle the first and last weeks of the year are admittedly obscure. I ask that you trust me on this one!

```
DATA DATETEST;
  SET DATETEST;

  * determine week start of datein;
  * determine the year it is in;
  * determine week start of Jan.1 that year;
  * determine day of week for 1/1 & 12/31;

WKDATEIN = INT((datein-2)/7)*7+2;
YRDATEIN  = YEAR(WKDATEIN);
WKJAN01 = INT((MDY(1,1,YRDATEIN)-2)/7)*7+2;
JAN01DAY = WEEKDAY(MDY(1,1,YRDATEIN));
DEC31DAY = WEEKDAY(MDY(12,31,YRDATEIN));

  * if 1/1 falls on Sun-Wed, it is in wk 1;
  * else it falls in last week of prev. year;
  * subsequent weeks are offset from this week;

IF (JAN01DAY GE 1 AND JAN01DAY LE 4) THEN
   WEEKNUM = (WKDATEIN-WKJAN01)/7+1;
ELSE
   WEEKNUM = (WKDATEIN-WKJAN01)/7;

  * create yyyynn form;

YEARWK = YEAR(WKJAN01+7)*100+WEEKNUM;

  * weeknum should be 1, not 53, if 12/31;
  * falls on Sun-Tues.  Adjust YEARWK;

IF (WEEKNUM = 53) AND (DEC31DAY GE 1 AND
    DEC31DAY LE 3) THEN DO;
      WEEKNUM = 1;
      YEARWK = (YRDATEIN+1)*100+WEEKNUM;
END;
```

Figure-7

Note that the WEEKOF calculation, described previously, was used here to facilitate determining the proper week number.

This algorithm is not pretty, but it does work correctly and it saves a tremendous amount of code and date maintenance in many of my programs. Again, for the parts that are not as straightforward, it is important to be liberal with comments.

Conclusion

I hope that this paper has enlightened you to the many ways that SASdates can be used, and has inspired you to try some of these ideas in your own programs. Once you get past the mechanics of how SASdates are created and used, the power of this sequential concept will become apparent.

I believe that the power of SASdates lies mainly in two areas: streamlined code and reduced code maintenance. As I have shown in the above examples, there are a wealth of opportunities for adding SASdates to programs. I believe that once you begin using them, you will amazed at the applicability of SASdates. And I believe that you will find significant time savings from reduced program maintenance. SASdates should be carefully examined and used to their fullest extent.

[1] SAS is a registered trademark of SAS Institute, Inc. Cary, NC USA.

[2] FOCUS is a registered trademark of Information Builders, Inc. New York, NY USA.

[3] LOTUS 1-2-3 is a registered trademark of Lotus Development Corp. Cambridge, MA USA.

Author

Kenneth Goodwin
Inland Steel Company
3210 Watling Street, MC 8-216
East Chicago, IN 46312
Phone 219/399-4938
Fax 219/399-4875
Email KLGOOD@INLAND.COM