Handling the Year 2000 and Other Timely Issues

David Beam
Systems Seminar Consultants
Madison, WI 53716 (608) 222-7081

Abstract

As we approach the year 2000, smart IS professionals are planning for a successful start of the next century. The SAS System® has some of the most powerful date handling capabilities of any software system, and can offer valuable tools to prepare your computer files for the end of the 90's.

One of the areas in which SAS Software excels is the powerful capabilities to do sequential date processing. This paper will present a brief overview of how SAS handles dates, and some examples of conversions required to read in a variety of dates. Examples of a several unique date calculation problems will also be presented, as well as a discussion of the year 2000 issues.

Introduction

The SAS method for doing date comparisons, math on date fields, sorting dates etc. is to use a numeric sequential representation of a date value. All date values in SAS can be converted to a numeric value representing the number of days from Jan. 1, 1960. Dates after this value are positive numbers, dates before this are negative numbers. (My birth date is a BIG negative number in SAS!).

We use a variety of *informats* and *formats* to read in these dates and display them correctly, and a wide variety of *functions* are available to manipulate the dates as needed. Times of day can also be represented as a numeric value inside SAS, converted to the number of seconds that has passed since midnight. A combination of a date and a time of day can also be used in SAS, internally the value is the number of seconds from midnight on January 1, 1960.

Sample of some dates/times in SAS:

What we see:	What SAS uses:
01/27/1997	13,541
01/27/1954	-2,165
06:45 AM	24,300
6:45 AM and 37 seconds	24,337
06:45 PM	67,500
06:45AM, 1/27/1997	1,169,966,700
06:45PM, 1/27/1954	-186,988,500

As you can see, these SAS numeric values are not what we want to see, so we use FORMATS to display them correctly.

A Brief List of some SAS Informats/Formats

This is by no means meant to be a comprehensive list, refer to the SAS documentation for complete lists:

Format/Informat:	Example:
MMDDYY8.	03/19/97
MMDDYY10.	03/19/1997
MMDDYY6.	031997
YYMMDD10.	1997-03-19
DATE7.	19MAR97
DATE9.	19MAR1997
DAY2.	19
DOWNAME.	Wednesday
WEEKDAY.	4
WEEKDATE17.	Wed, Mar 19, 1997
MMYYs7.	03/1997
MONYY7.	MAR1997
JULIAN5.	97078
JULIAN7.	1997078

A Brief List of some SAS date/time Functions

Again, this is a partial list of some of the SAS functions used to manipulate dates and times of day:

Function:	Description:
DATE	Returns current date
DATEJUL	Convert a Julian date to a SAS date
DAY	Returns the day of month from a SAS date
HMS	Return SAS time from hours, minutes, seconds
HOUR	Return hour from SAS time or datetime
INTCK	Return number of intervals between dates
INTNX	Advance date/time by an interval
JULDATE	Return Julian date from a SAS date
MDY	Return a SAS date from month, day, year
MONTH	Return month from SAS date
QTR	Returns calendar quarter from SAS date
TIME	Return current time of day
WEEKDAY	Returns day of week (Sunday=1, Monday=2,...Saturday=7)
YEAR	Return year from SAS date
YYQ	return SAS date from year and quarter

Reading in a few dates:

The following examples show how to read in a few dates from a non-SAS data file by using a combination of SAS informats and functions. The resulting variables will all be SAS numeric date values.

Example 1:

Reading a DB2 style of date, coming in as 1997-03-19:

```
INPUT  @1 bdate  YYMMDD10.;
```

Results in a numeric value 13,592.

Example 2:

Suppose you have a COBOL created file, with the date stored as a Julian date but in "packed decimal" on the file. This requires two SAS statements, to read in the Julian date, and then to convert it with a function:

```
INPUT @1 bdate PD3.;
        /* result is value 97078 */

bdate = DATEJUL(bdate);
        /* result is value 13,592 */
```

Example 3:

Now suppose the date is a numeric value, in packed format, but the value in the input field contains month, day and year as 31997, for March 19, 1997. (Oct. 23, 1996 would be 102396).

This must be read in as a numeric value and converted to a SAS date value by first creating a character string and passing that through the correct date informat with the INPUT function:

```
INPUT @1 bdate PD4.;
        /* result is value 31997 */

datec = PUT(bdate, $6.);
bdate = INPUT(datec, MMDDYY6.);
        /* result is value bdate = 13,592 */
```

The above two statements can be nested, as:

```
bdate = INPUT( PUT(bdate,$6.), MMDDYY6.);
```

Example 4:

In the worst case, if you can't use any informat or function directly to convert your date to a SAS date value, try to get the month, day and year of the date coming in as numeric values. Then the MDY function can be used to create the SAS date value:

```
INPUT   @1      month   2.      /* month = 03 */
        @65     year    4.      /* year = 1997 */
        @97     day     2.;     /* day = 19 */

BDATE = MDY(month, day, year);
        /* results is value of BDATE = 13592 */
```

Hard Coding SAS Dates or Times

Often in a SAS program you will need to type in a value of a date, time or datetime for comparison or computing with dates. The Date, Time and DateTime constants as shown below allow 'hard-coding' of dates and times:

```
'19MAR1997'D    ==> date constant
'14:45'T        ==> time constant
'19MAR1997:14:45'DT
                ==> DateTime constant
```

Now the following code can be used, for example, to select all invoices between two date ranges. This assumes, of course, that the variable INV_DTE already contains a numeric SAS date value!

```
WHERE  INV_DTE  GE  '15FEB1997'D
       AND  INV_DTE  LE  '28FEB1997'D;
```

Suggestion: When using date constants, code the complete four-digit year, to prevent end-of-century problems. We really are close to the year 2000! (More on this topic later.)

Using Simple SAS Date Functions

Most of the SAS date/time functions are easy to use and understand. The following functions all return a numeric value when a SAS date value is passed as the argument (with 19MAR1997 as the current date):

```
Today = DATE();             /* today's date as numeric */
                            /* returns 13592) */

Year = YEAR(today);         /* returns: 1997 */
Julian = JULDATE(today);    /* returns 97078 */
Quarter = QTR(today);       /* returns: 1 */
Month = MONTH(today);       /* returns: 3 */
Day = DAY(today);           /* returns: 19 */
Wday = WEEKDAY(today);
                            /* returns: 4 ( Wednesday )*/
```

Interval Functions

Two very special date/time functions need to be discussed. These are used to compute a date/time in the future or past, or compute how many of a specified interval passed between two dates or times.

The 'Interval Check' Function:

INTCK('*interval', from, to*)

Returns the number of time intervals in a given span. The count is how many of the specified interval **began** between the dates.

The 'Interval Advance' Function:

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

Advances a *from* date/time value by the specified *number* of *interval*s, to the date/time at the requested *alignment*. If no alignment given, the date/time is advanced to the **beginning** of the specified interval.

(The *'alignment'* argument is **new with release 6.11** and will cause an error if used in earlier versions of SAS.)

In both of the above, the *interval* argument is a character string representing the requested interval:

Date intervals	Time intervals	Datetime intervals
WEEK	HOUR	DTDAY
MONTH	MINUTE	DTWEEK
QTR	SECOND	DTMONTH
YEAR		DTQTR
DAY		DTYEAR

The following intervals were added with **release 6.07**, for both Date and DateTime intervals:

WEEKDAY	TENDAY	SEMIMONTH
SEMIYEAR		

Release 6.11 added the optional fourth argument to the **INTNX** function, to allow specification of the 'alignment' of the interval to advance to. This argument can have the following values:

'BEGINNING' (Default)	or 'B'	beginning of the interval
'MIDDLE'	or 'M'	midpoint of the interval
'END'	Or 'E'	returns end of interval

Example 1
Using the INTCK function with NOW = '19MAR1997'D:

```
INTCK('WEEK', now, '21MAR1997'D);
      returns 0 (same week)
INTCK('WEEK', now, '24MAR1997'D);
      returns 1(next week)
INTCK('MONTH', now, '30MAY1997'D);
      returns 2(two weeks in future)
INTCK('MONTH', now, '23MAR1997'D);
      returns 0 (same month)
INTCK('TENDAY', now, '01APR1997'D);
      returns 2 (2-ten day intervals)
INTCK('SEMIYEAR', now, '31DEC1997'D);
      returns 1 (half a year)
```

Example 2
Using the INTNX function with NOW = '19MAR1997'D

(SAS returns a SAS date value, shown here in the MMDDYY8. format):

```
INTNX('WEEK', now, 1);
      returns 03/23/97 (next Sunday)
INTNX('WEEK', now, 2);
      returns 03/30/97 (a week from Sunday)
INTNX('WEEK', now, 1, 'E');
      returns 03/29/97 (this Saturday)
INTNX('WEEK', now, 1, 'M');
      returns 03/26/97 (next week Wednesday)
INTNX('MONTH', now, 0);
      returns 03/01/97 (start of this month)
INTNX('MONTH', now, 0, 'END');
      returns 03/31/97 (last day of current month)
```

How to Get Today's Date in a Title

There is a SAS system macro variable, *&SYSDATE*, that will return today's date as a character string, in the SAS *DATE7.* format. For example:

```
TITLE "This was run on &SYSDATE";
```

will resolve to a title:

This was run on 19MAR97

NOTE: Be sure to use double quotes in the TITLE, the *&sysdate* will **not** resolve if single quotes are used.

What if you don't like the style SAS uses for this date? Simple...create your own macro variable:

```
data _null_;

      /* get today's date as character string */
      CH_DATE = "&SYSDATE";

      /* convert to numeric date value */
      NU_DATE = INPUT(CH_DATE, DATE7.);

      /* put to character string in desired date format */
      MY_DATE = PUT(NU_DATE, MMDDYY10.);

      /* create MACRO variable holding formatted date */
      CALL SYMPUT("MYDATE", MY_DATE);
run;

title "This was run on &MYDATE";
      resolves to:    This was run on 03/19/1997
```

NOTE: above statements could be nested to on one CALL SYMPUT:

```
DATA _null_;
```

```
    CALL SYMPUT("mydate",
         PUT(INPUT("&sysdate",DATE7.),MMDDYY10.));
RUN;
```

Real-World Example 1

The following shows where date manipulation in SAS can really help. Suppose your department is responsible for tracking the performance of vendors. The boss wants to know how long it takes, in hours, for a vendor to respond to a question.

If you record the date and time when you call the vendor, and the date and time when they call you back, you can subtract the two values and compute the response time in hours:

```
data vendors;
  input @1   called      datetime13.
        @15 answer    datetime13.
        ;
  time = answer - called;
  hours = (time / 60) / 60;
  hours = round(hours, .01);
  format called answer datetime13.;
datalines;
19MAR97:08:45 19MAR97:14:22
19MAR97:08:45 20MAR97:14:22
;
run;
```

This produces the following:

CALLED	ANSWER	HOURS
19MAR97:08:45	19MAR97:14:22	5.62
19MAR97:08:45	20MAR97:14:22	29.62

As can be seen, the hours for the second example includes night time. So now the boss says "Don't include the time between 5 PM and 8 AM, as our office is closed."

Change the program as follows (changed code is highlighted):

```
data vendors;
  input  @1   called      datetime13.
         @15 answer    datetime13.
       ;
  time = answer - called;
  /* see if overnight, DATEPART gets date alone */
  if DATEPART(answer) GT DATEPART(called) then
    do;
      /* remove 5-12PM and 12PM-8AM hours */
      /*   or 15 hours for each day                */
      days = INTCK('DTDAY', called, answer);
      time = time - (15 * 60 * 60 * days);
    end;
  hours = (time / 60) / 60;
  hours = round(hours, .01);
  format called answer datetime13.;
datalines;
19MAR97:08:45 19MAR97:14:22
19MAR97:08:45 20MAR97:14:22
;
run;
```

Results:

CALLED	ANSWER	HOURS
19MAR97:08:45	19MAR97:14:22	5.62
19MAR97:08:45	20MAR97:14:22	14.62

Ok, almost perfect. But what about the weekends?

Now change the program to remove 24 hours for each day that falls over a weekend. The INTCK function for the 'WEEK' interval shows us how many Sundays pass. Take that count times 2 days.

New program:

```
data vendors;
  input  @1   called      datetime13.
         @15 answer    datetime13.
        ;
  time = answer - called;
  /* see if overnight, DATEPART gets date alone */
  if DATEPART(answer) GT DATEPART(called) then
    do;
      /* remove 5-12PM and 12PM-8AM hours */
      /*   or 15 hours per day           */
      days = INTCK('DTDAY', called, answer);
      time = time - (15 * 60 * 60 * days);
    end;
  /* check for weekends, remove an extra 9 hours */
  /*   for each of the two weekend days          */
  weeks = INTCK('DTWEEK', called, answer);
  if weeks > 0 then
    do;
      time = time - (9 * 2 * 60 * 60 * weeks);
    end;
  hours = (time / 60) / 60;
  hours = round(hours, .01);
  drop days time weeks;
  format called answer datetime13.;
datalines;
19MAR97:08:45 19MAR97:14:22
19MAR97:08:45 20MAR97:14:22
19MAR97:08:45 01APR97:08:10
;
run;
```

and the final results:

OBS	CALLED	ANSWER	HOURS
1	19MAR97:08:45	19MAR97:14:22	5.62
2	19MAR97:08:45	20MAR97:14:22	14.62
3	19MAR97:08:45	01APR97:08:10	80.42

Testing these kinds of date and time logic problems can be done by manually comparing the results with what SAS computed. Look at a calendar and compare the results above.

Real-World Example 2

We had a need to count how many times each of the 12 months occurred between two dates, that is, how many Januaries, how many Februaries etc. occurred between two dates.

This problem is simple to solve in SAS but requires a little thinking. The following program computed 12 variables to hold the count of how many of that month passed. This example counts the starting month once and the ending month once.

The program:

```
 /* Count how many of each of the 12 months occur */
 /*   between 2 dates.                            */

data test;
 start='19MAR1997'D;
 input @1 end mmddyy10.;

 /* set up 12 variables to count each month, */
 /*   set all to zero. */
 length m1-m12 4;
 array cmonths {12}  m1-m12;
 do i = 1 to 12;
   cmonths{i}=0;
 end;

 /* grab starting month number */
 i_month=month(start);

 /* Compute number of months between dates (the +1 */
 /*   adds one for the starting month)        */
 months=intck('month',start,end)+1;

 /* For each month between start and end, add 1 to the */
 /*   appropriate monthly counter.                     */
 do i = 1 to months;
   cmonths{i_month}= cmonths{i_month}+1;
   i_month+1;
   /* After December, reset to January */
   if i_month = 13 then i_month = 1;
 end;

 drop i_month i months start;
 format end mmddyy8.;

datalines;
03/21/1997
04/01/1997
06/03/1997
09/01/1997
09/30/1997
03/01/1998
01/01/1998
05/05/1998
;
run;

options ls=77;
title "How Often Each Month Occurs from 19MAR1997";
proc print label;
 label m1='Jan' m2='Feb' m3='Mar' m4='Apr' m5='May' m6='Jun'
       m7='Jul' m8='Aug' m9='Sep' m10='Oct' m11='Nov' m12='Dec';
run;
```

The Results:

```
  How Often Each Month Occurs from 19MAR1997

   E      J   F   M   A   M   J   J   A   S   O
N   D
     N      a   e   a   p   a   u   u   u   e   c
 o   e
     D      n   b   r   r   y   n   l   g   p   t
 v   c

21MAR97   0   0   1   0   0   0   0   0   0   0
 0   0
01APR97   0   0   1   1   0   0   0   0   0   0
 0   0
03JUN97   0   0   1   1   1   1   0   0   0   0
 0   0
01SEP97   0   0   1   1   1   1   1   1   1   0
 0   0
30SEP97   0   0   1   1   1   1   1   1   1   0
 0   0
01MAR98   1   1   2   1   1   1   1   1   1   1
 1   1
01JAN98   1   0   1   1   1   1   1   1   1   1
 1   1
05MAY98   1   1   2   2   2   1   1   1   1   1
 1   1
```

Real-World Example 3

Another date logic problem we've encountered is the need to compute a SAS date that is the same day of the month for the prior month.

For example, if today is 03/19/1997, the same day last month was 02/19/1997. The difficult dates are the ends of months when the prior month is shorter, for example, if today is 03/30/1997, what is the same day last month when it doesn't exist? You could make a case for 02/28/1997 or 03/02/1997. This solution forces the above to 02/28/1997:

```
DATA TEST;
  /* get start date as sas date value */
  INPUT  @1  NOW_DATE  MMDDYY10.;

  /* DEFAULT = SAME DAY LAST MONTH    */
  OLD_DAY   = DAY(NOW_DATE);
  OLD_MON   = MONTH(NOW_DATE) - 1;
  OLD_YEAR  = YEAR(NOW_DATE);
 /* if now is January, last month Dec prior
year */
  IF MONTH(NOW_DATE) = 1 THEN
    DO;
      OLD_MON = 12;
      OLD_YEAR  = YEAR(NOW_DATE) - 1;
    END;

  OLD_DATE = MDY(OLD_MON, OLD_DAY, OLD_YEAR);

  /* NOW ALLOW FOR LAST MONTH SHORTER DAYS
THAN   */
  /*   CURRENT MONTH (EXAMPLE: MARCH 31, SAME
DAY */
  /*   LAST MONTH IS EITHER FEB. 28 OR FEB.
29!)  */
  IF OLD_DATE = . THEN         /* . INVALID
DATE  */
   DO;
    OLD_DATE = INTNX('MONTH',NOW_DATE,0) - 1;
   END;
  KEEP NOW_DATE OLD_DATE;
  FORMAT NOW_DATE OLD_DATE MMDDYY10.;
```

```
DATALINES;
12/31/1996
01/31/1997
03/31/1997
03/29/1996
03/30/1996
03/31/1996
;
RUN;

PROC PRINT DATA=TEST LABEL;
  TITLE "COMPUTING SAME DAY LAST MONTH";
  LABEL OLD_DATE = "SAME DAY LAST MONTH";
RUN;
```

The Results:

```
COMPUTING SAME DAY LAST MONTH

                       SAME DAY
OBS      NOW_DATE     LAST MONTH

 1     12/31/1996    11/30/1996
 2     01/31/1997    12/31/1996
 3     03/31/1997    02/28/1997
 4     03/29/1996    02/29/1996
 5     03/30/1996    02/29/1996
 6     03/31/1996    02/29/1996
```

The Year 2000 - Be Careful!

Many non-SAS files exist with dates stored in a variety of methods. Those without the full four-digit year will need to be modified or the programs reading those dates may make incorrect assumptions about the century.

If you are working with data that only has a two digit year, as in "02/15/37", you need to be aware of how that converts in SAS. The informat "MMDDYY8." will only see the year as 37, and SAS needs to make an assumption as to whether this is 1937 or 2037.

A SAS global option called "YEARCUTOFF=" is available to help control this situation. The YEARCUTOFF= option tells SAS to use the current century or the next century when working with only a two digit year. If you have only a two digit year, SAS looks at the last two digits of the YEARCUTOFF= option.

If the two digit year is LESS than the last two digits of the YEARCUTOFF= setting, SAS assumes the NEXT century.

If the two digit year is the SAME or GREATER than the last two digits of YEARCUTOFF=, SAS assumes the year is the same century as YEARCUTOFF=.

For example, let's assume that the YEARCUTOFF= has been set as follows:

. . **OPTIONS YEARCUTOFF=1950;**

Your two digit years will be interpreted as follows:

Input Date	**SAS Translates to:**
01JAN49	January 1, 2049
01JAN50	January 1, 1950
01JAN51	January 1, 1951
01JAN05	January 1, 2005
04/25/00	April 25, 2000
12/15/1949	December 15, 1949

(If a full 4 digit year is read in, that will be used by SAS).

The ***default*** from SAS is...

. . **OPTIONS YEARCUTOFF=1900;**

... which results in ***ALL*** two digit years falling into the **1900s**.

WARNING!!

The YEARCUTOFF option is NOT supported under versions of SAS prior to release 6. If you are still running version 5, here's another good reason to upgrade those programs!!

NOT A MAGIC BULLET...

The above SAS default is NOT the cure-all end-all solution, any time a computer makes an assumption about something it could be wrong!! Birth dates are a very simple example of how the above option may not always work. In the first example above, the 01JAN49 as a birth date would compute 2049 as birth year. Getting younger may be nice in some regards...

Several techniques have been created by a variety of companies to handle mass conversion of dates. In a nut shell, the dates with 2-digit year values need to be expanded to 4-digits to be truly safe. The hardest part of changing these file layouts to expand the date field is not the conversion step, but the communication and timing with the users of those files.

These users may be other systems or programs, or staff members doing ad-hoc reporting on the files. In companies that allow end-users freedom to read files as needed, this is a huge task to change these files. Standardized SAS program code that handles the input of non-SAS data is a step in the right direction, but this is not the case in many organizations.

Some of the common date problems may be programs that assume '99' is an invalid year, or retention or expiration dates of 9/9/99 as the oldest date that some event can happen.

As of March 19,1997 it is only **1,018** days until 01/01/2000!

ONE TECHNIQUE TO MODIFY DATES IN NON-SAS FILES

Using an External package, such as Syncsort

One example we have seen under the MVS operating system uses *Syncsort* to read in a file and write it out again, adding the '19' prefix to date fields. This works for fixed length records, with the date field as either character or packed decimal. If the packed field has the 2-digit year in a single byte, such as a Julian date in the form YYDDDs (s=sign), this can be output with a '19' value added to the front. For dates in different formats, SAS programs can be developed that can be easily adapted to a variety of date styles.

The following is a sample of a fixed length (record length=40) file with two date fields needing the '19' prefix. The first is at offset 21 as a character date in the form YYMMDD, the second a 3-BYTE packed-decimal Julian date at offset 31, in the form YYDDD:

```
// EXEC SYNCSORT
//SORTIN    DD DSN=file in
//SORTOUT   DD new file out (with new record size)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,20,         FIRST 20 COLS
                21:C'19',     INSERT "19"
               23:21,10,     FIRST DATE+NEXT 4 BYTES
                33:X'19',     INSERT "19"
                34:31,10)     REST OF DATA
 END
/*
```

USING SAS CODE TO PERFORM ABOVE CHANGES

```
 DATA _NULL_;
  INFILE whatever-in;
   INPUT   @1   CHUNK1     $CHAR20.
          @21  CHUNK2     $CHAR10.
         @31  JULIAN     PD3.
        @34  CHUNK3     $CHAR7.
       ;
JULIAN = 1900000 + JULIAN;
   FILE whatever-out;
   PUT     @1   CHUNK1     $CHAR20.
          @21 '19'
         @23  CHUNK2     $CHAR10.
        @33 JULIAN     PD4.
       @37  CHUNK3     $CHAR7.
     ;                                    RUN;
```

The above is a simple example of reading an external file and writing out a new file with changes to the date fields. The layouts can be adapted for a variety of situations, SAS is a great tool for rewriting files to a new layout.

Year 2000 Trivia

1. Year 2000 problems can only occur on or after the first day of year 2000.

   True False

2. Which years in the following list are leap years?

   1900, 1982, 1988, 1990, 2000, 2004

1) False. These problems have already occurred since so much data is stored with only 2 digit years. Trust me.

2) 1988, 2000 and 2004 are leap years. Common misunderstandings about leap year logic are handled quite nicely is SAS.

Who's in Charge?

The Gartner Group has issued the following Strategic Planning Assumption: Without corrective measures, 90% of business applications will fail by 1999 due to invalid date computations (0.8 probability). Note that they said by 1999 not 2000!

Here are some of the reasons people may be ignoring the problem:
1. It's someone else's problem.
2. Someone smarter than you will come up with an automated solution.
3. You're planning to retire next year.
4. You just don't have the time right now. Ask again next year when things slow down.
5. You're too busy writing new applications.
6. January 1st 2000 falls on a Saturday and Monday's a holiday ..you'll have lots of time over the weekend.
7. Your standards (found in a large red binder) already outlines how dates should be used.
8. You only use vendor software. They'll take care of it.
9. You'll have replaced your application by then.
10. You don't have the budget.
11. You can't believe 2 missing digits can cause that much trouble.
12. There are no date problems in your code (you have faith in your programmers).
13. You're waiting for your management to ask you to work on it.
14. What date problem?
15. Bill Gates will solve it.

More reasons to ignore the problem:
1. You want to surprise the stockholders.
2. You lost the source code of your application 3 years ago.
3. Government will pass legislation to roll back the clock to 1900.
4. You're moving your mission critical systems to client server.

5. You're planning a vacation that week.
6. You just plan to have your programmer wear a beeper that week.
7. Computers have no impact on your life.
8. You believe in the Tooth Fairy.

Year 2000 Resources

The following is a list of resources the author has come across in the past year concerning the YR2K issue. There is no implied worth in these listings, nor can the author assume responsibility for any of the content.

IBM

IBM has published a manual, “Year 2000 and 2-digit Dates: A Guide for Planning and Implementation.” It is available thru your IBM rep or via the Internet at:

www.software.IBM.com/year2000/index.html

Topics covered include identifying 2-digit year exposures, reformatting year-date notation, testing techniques, migration considerations, tool (IBM's and other's) and a listing of Year 2000 ready program products and hardware. It even covers some of the PC problems and testing procedures to make sure your PC survives into Year 2000.

World Wide Web

Year 2000 Information Center
www.year2000.com

Information Technology Association of America
www.itaa.org/year2000.htm

CONCLUSION

Understanding dates and times and how SAS handles them can help solve many business analysis problems. It is easy to set up a sample test program as shown above, to test your calculations.

The year 2000 presents challenges that ALL computer users should be preparing for, now.

The author will be glad to answer questions and accept suggestions at the following address:

David Beam
Systems Seminar Consultants
1220 Femrite Drive
Suite 206
Madison, WI 53716

voice: (608) 222-7081
Internet: train@sys-seminar.com