CHAPTER 1

Introduction to Dates and Times in SAS

In the years that I've been working with SAS, and teaching students how to use it, I've noticed two things about it that consistently confuse programmers who are new to SAS. First, there is the "implied" DO-UNTIL (end-of-file) of the DATA step, and then there is the concept of dates (and times) within SAS. I've seen many misuses of character strings masquerading as dates and/or times over the past years. However, this is only the tip of the iceberg when it comes to the power and flexibility of dates and times in SAS. There is much more than just having numbers representing date and time values in SAS. We'll start with the basics in the first three chapters, and then progress to some more advanced uses of those date and time values.

1.1 How Does It Work? (January 1, 1960 and Midnight as Zero)

SAS has three separate counters that keep track of dates and times. The date counter started at zero on January 1, 1960. Any day before 1/1/1960 is a negative number, and any day after that is a positive number. Every day at midnight, the date counter is increased by one. The time counter runs from zero (at midnight) to 86,399.9999, when it resets to zero. The last counter is the datetime counter. This is the number of seconds since midnight, January 1, 1960. Why January 1, 1960? One story has it that the founders of SAS wanted to use the approximate birth date of the IBM 370 system, and they chose January 1, 1960 as an easy-to-remember approximation.

Many database programs maintain their dates as a value relative to some fixed point in time. This makes calculating durations easy, and working with dates stored in this fashion becomes a matter of addition, subtraction, multiplication, and division.

1.2 Internal Representation (Storage as Integers or Real Numbers)

SAS stores dates as integers, while the datetime and time counters are stored as real numbers to account for fractional seconds. The origin of the algorithm used for SAS date processing comes from a *Computerworld* article dated January 14, 1980 by Dr. Bhairav Joshi of SUNY-Geneseo. The earliest date that SAS can handle with this algorithm is January 1, 1582. The latest date is far enough into the future that four digits can't display the year.

1.3 External Representation (Basic Format Concepts)

The dates as stored by SAS don't do us much good in the real world. The statement "I was born on –242" won't mean much to anyone else. On the other hand, "May 4, 1959" can easily be translated into something that most people can understand. SAS has a built-in facility to perform automatic translation between SAS numbers and dates and times as understood by the rest of the world. This automatic translation is performed with what are called formats. Formats display the date, time, and datetime values in a fashion that is much more easily understood. Formats do *not* change the values themselves; they are just a way to display the values in any output.

What happens if you have a date or time and want to translate it into SAS date and time values? SAS has another built-in facility which performs the reverse translation, from the dates and times we understand and use to the values that SAS stores. This translation is done using informats. Informats translate what they are given into the values that are stored in SAS variables. We will discuss formats and informats in detail in Chapters 2 and 3, because there are dozens of them.

1.4 Date and Time as Numeric Constants in SAS

We've talked about internal and external representation of dates and times. How do you put a specific date into a program as a constant? Formats change the way the values are displayed in output, so you can't use them. Informats translate what they are given, so you could use them, but then you'd need to use the INPUT() function (see Section 3.3.2), which takes a value you give it and translates it with an INFORMAT. That's very inefficient. Look at the following program (Example 1.4.1) to see how date, time, and datetime constants are written into a SAS program. Take note of the quotation marks around the values for date, time, and datetime, and the letters that follow each closing quote.

The quotes are used to create a literal value. You may use a pair of single or double quotes to specify the literal value. The only difference between using single and double quotes around the date would be macro expansion. The most important part of a date constant is the letter that immediately follows the last quote. The letter "D" stands for date, "T" for time, and "DT" for datetime, and you can use either upper or lowercase. If you put a date in quotes without the letter at the end, you will create a character variable, not a numeric variable with a date, time, or datetime value. The difference might not become apparent until you try to do something with the variable you created that involves a calculation. Don't forget your "D", "T", or “DT”! Example 1.4.1 demonstrates how date constants are defined and then automatically converted to SAS date values.

Example 1.4.1 Date Constants

```
DATA date_constants;

date = ‘04aug2004’d;  /* This is a date constant */
time = ‘07:15:00’t;   /* This is a time constant */
datetime = ‘07aug1904:21:31:00’dt;  /* This is a datetime constant */
run;

TITLE “Unformatted Constants”;
PROC PRINT DATA=date_constants;
VAR date time datetime;
run;

TITLE “Formatted Constants”;
PROC PRINT DATA=date_constants;
VAR date time datetime;
FORMAT date worddate32. time timeampm9. datetime datetime32.;  /* Format
the constants */
run;
```

Here is the resulting output:

Unformatted Constants		
date	**time**	**datetime**
16287	**26100**	**-1748226540**

Formatted Constants		
date	**time**	**datetime**
August 4, 2004	**7:15 AM**	**07AUG1904:21:31:00**

Without formats, you can see that the date constants we created are stored as their actual SAS date, time, and datetime values. They don’t make much sense until you format them.

1.5 System Options Related to Dates

SAS has several system options; these affect the way that the SAS job or session works. There are four important options that affect dates: YEARCUTOFF, DATESTYLE, DATE/NODATE, and DTRESET.

YEARCUTOFF

IMPORTANT

On December 31, 1999, people were holding their breath. The majority of dates stored on computers allowed only two digits for the year, and assumed that the first two digits were (and would always be) "19". This didn't account for storage of dates where the first two digits of the year were not "19", and thus, the "Y2K problem" was born. How does SAS handle two-digit years? When is a two-digit year in the 1900's, and when is it in the 2000's? What if you have old data and all those dates need to be in the 1800's? What does SAS do? The answer is: *YOU* tell SAS how to handle two-digit years. There is a system option called YEARCUTOFF that lets you specify a 100-year span for two-digit years. It applies to all dates with two-digit years that you give SAS. This means that it applies to: date constants, date values read from raw data with the INPUT statement, and date values that are created from character strings with the INPUT() function. The YEARCUTOFF system option does not affect values that are stored as SAS date values, regardless of their display, so once you create a date or datetime value, YEARCUTOFF no longer has any effect on it.

The system default is 1920. This means that any two-digit year from 20 to 99 will be translated as 1920 to 1999, while years from 00 to 19 will be translated as 2000 to 2019. The syntax is:

```
OPTIONS YEARCUTOFF= (y)yyyy;          /* (y)yyyy can be from 1582 to 19900 */
```

Let's use a series of OPTIONS statements and date constants to illustrate. In the following program, three datasets are created with four identical date constants that use two-digit years. The only thing that changes is the value of YEARCUTOFF. Example 1.5.1 shows how YEARCUTOFF translates two-digit year values using date constants.

Example 1.5.1 How the YEARCUTOFF System Option Works

```
OPTIONS YEARCUTOFF=1920;  /* SAS System default */

DATA yearcutoff1;
date1 = "15JUL06"d;
date2 = "27FEB48"d;
date3 = "04may69"d;
date4 = "10dec95"d;
RUN;

PROC PRINT DATA=yearcutoff1;
FORMAT date1-date4 mmddyy10.;
RUN;
```

Here is the resulting output:

date1	date2	date3	date4
07/15/2006	02/27/1948	05/04/1969	12/10/1995

With the default of 1920 in effect, you can see that the first date is placed in the 21st century, while the others remain in the 20th. Let's move the 100-year period back by 80 years and see what happens.

```
OPTIONS YEARCUTOFF=1840;

DATA yearcutoff2;
date1 = "15JUL06"d;
date2 = "27FEB48"d;
date3 = "04may69"d;
date4 = "10dec95"d;
RUN;

PROC PRINT DATA=yearcutoff2;
FORMAT date1-date4 mmddyy10.;
RUN;
```

Here is the resulting output:

date1	date2	date3	date4
07/15/1906	**02/27/1848**	**05/04/1869**	**12/10/1895**

Now the first date is in the 20th century, and the others are in the 19th. Note that the only change to the code is in the OPTIONS statement. The value of YEARCUTOFF is 1840 instead of 1920. For the last part of this example, we'll set YEARCUTOFF to 1970, and use the same date constants with two-digit years again.

```
OPTIONS YEARCUTOFF=1970;

DATA yearcutoff3;
date1 = "15JUL06"d;
date2 = "27FEB48"d;
date3 = "04may69"d;
date4 = "10dec95"d;
RUN;

PROC PRINT DATA=yearcutoff3;
FORMAT date1-date4 mmddyy10.;
RUN;
```

Here is the resulting output:

date1	date2	date3	date4
07/15/2006	**02/27/2048**	**05/04/2069**	**12/10/1995**

Once again, the only difference in the code is in the OPTIONS statement. Now the 100-year range starts in 1970, which places every date except the last one in the 21st century.

As with many SAS system options, YEARCUTOFF is effective when it is encountered within the program. If you have multiple OPTIONS statements that include YEARCUTOFF= in your program, each one will affect all date constants, raw data, and date values created from character strings with the INPUT() function until the next OPTIONS YEARCUTOFF= statement changes the 100-year range. As an example, if you were to put the three programs in the above example together in one file, the result would be the same, as long as you did not move the OPTIONS YEARCUTOFF= statements.

DATESTYLE

This system option is important if you are using any of the following informats: ANYDTDTE., ANYDTDTM., or ANYDTTME. DATESTYLE controls how SAS will translate dates that can be interpreted in more than one way. This happens most often when you are using two-digit years.

Assuming that the OPTIONS statement specifies YEARCUTOFF=1920, does 11-01-06 mean November 1, 2006, January 6, 2011, or January 11, 2006?

DATESTYLE allows you to tell SAS how to interpret cases like this. You may specify any one of the following:

Table 1.5.1 *Values for DATESTYLE=*

MDY	Sets the default order as month, day, year. "11-01-06" would be translated as November 1, 2006	**YDM**	Sets the default order as year, day, month. "11-01-06" would be translated as June 1, 2011
MYD	Sets the default order as month, year, day. "11-01-06" would be translated as November 6, 2001	**DMY**	Sets the default order as day, month, year. "11-01-06" would be translated as January 11, 2006
YMD	Sets the default order as year, month, day. "11-01-06" would be translated as January 6, 2011	**DYM**	Sets the default order as day, year, month. "11-01-06" would be translated as June 11, 2001
LOCALE (default)	Sets the default value according to the LOCALE= system option. When the default value for the LOCALE= system option is "English_US", this sets DATESTYLE to MDY. Therefore, by default, "11-01-06" would be translated as November 1, 2006.		

DATESTYLE can be set at SAS invocation, through an OPTIONS statement, in the configuration file, or in the SAS Options window. The syntax is:

OPTIONS DATESTYLE=*order*; **/* order is one of the values from table 1.1 */**

Example 1.5.2 demonstrates the effect of the different DATESTYLE values on a given character string.

Example 1.5.2 How DATESTYLE Affects the ANYDTDTE. Informat

The following program goes through each of the possible values for DATESTYLE using the same character string 11-01-06 as input. The log shown below the program will demonstrate the differences.

```
OPTIONS DATESTYLE=mdy;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=mdy, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;

OPTIONS DATESTYLE=myd;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=myd, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;

OPTIONS DATESTYLE=ymd;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=ymd, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;

OPTIONS DATESTYLE=ydm;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=ydm, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;
```

```
OPTIONS DATESTYLE=dmy;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=dmy, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;

OPTIONS DATESTYLE=dym;
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=dym, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;

OPTIONS DATESTYLE=locale; /* LOCALE=EN_US */
DATA _NULL_;
INPUT date anydtdte8.;
PUT "OPTIONS DATESTYLE=locale, so date=" date mmddyy10.;
DATALINES;
11-01-06
;
RUN;
```

The Log

```
OPTIONS DATESTYLE=mdy, so date=11/01/2006
OPTIONS DATESTYLE=myd, so date=11/06/2001
OPTIONS DATESTYLE=ymd, so date=01/06/2011
OPTIONS DATESTYLE=ydm, so date=06/01/2011
OPTIONS DATESTYLE=dmy, so date=01/11/2006
OPTIONS DATESTYLE=dym, so date=06/11/2001
OPTIONS DATESTYLE=locale, so date=11/01/2006
```

As you can see, DATESTYLE can have an enormous effect when the ANYDTDTE. (or ANYDTDTM. or ANYDTTM.) informats are used.

DATE/NODATE

By default, the DATE system option is in effect when you start SAS, which causes the date and time that the SAS job (or session) started to appear on each page of the SAS log and SAS output. These values are obtained from the operating system clock. If you are running SAS interactively, then the date and time are printed only on the output, not the log. If you don't want the date and time to appear, use the NODATE system option. The syntax is:

```
OPTIONS NODATE;
```

If you've turned off DATE, then you can turn it back on with:

```
OPTIONS DATE;
```

Example 1.5.3 shows what happens to the title line printed by SAS when you use DATE and NODATE. Remember that, by default, DATE is in effect when you start SAS.

Example 1.5.3 DATE/NODATE

This is a sample of a title line with the DATE system option:

```
The SAS System         17:20 Thursday, August 5, 2004   1
```

This is what NODATE does to that title line:

```
The SAS System                                          1
```

DTRESET

If the DATE option is enabled, SAS prints the date and time that the current SAS session started. If you want a more accurate date and time on those pages, you can use the DTRESET system option. This will cause SAS to get the date and time from the operating system clock each time a page is written. That date and time will then be put on the page instead of the time that the SAS job started. Since the time is displayed in hours and minutes, you will see it change each minute only. The syntax is:

```
OPTIONS DTRESET;
```

1.6 Length and Numeric Requirements for Date, Time, and Datetime

Since dates are stored as integers, you can take advantage of that to save space when you create variables to store them. Instead of using the default length of 8 for numeric variables, set the LENGTH of the numeric variables where you are storing the dates to 4. This will safely store dates from January 1, 1582 (the earliest date SAS can handle), to October 23, 7701. A length of 5 is overkill, although that would extend the ending date another 534,773,760 days! A length of 3 will not accurately store dates outside the range of January 1, 1960 and September 13, 1960. If you declare your date variables to be a length of 4, you will be able to store two dates in the space it would take to store one if you were using the SAS default length for numeric variables.

Times may present a little bit of a problem, since times have the possibility of having decimal parts. You can get away with storing times in the same magic length of 4 that you can use for dates, and the rule is simple enough: if you want fractional seconds in your time values, use a length of 8 for maximum precision. Otherwise, the same length of 4 will store every possible whole second from midnight to midnight.

Datetime values need to be a little longer; a length of 4 will not store a datetime value with accuracy, regardless of whether you want decimal places. The number is just too big. Use a length of 6 to store datetime values; this will accurately represent datetime values (without fractions of seconds) from midnight, January 1, 1582 to 3:04:31 PM on April 9, 6315. Note that a length of 6 might not translate into other databases.

CAUTION

In all the above cases, the minimum lengths for accuracy have been given to you; do not attempt to save more space by shrinking the variables further. You will lose precision, and this could lead to unexpected results. Example 1.6.1 shows what can happen if you do not use enough bytes to store your date values.

Example 1.6.1 The Effect of LENGTH Statements on Dates

```
DATA date_length;
LENGTH len3 3 len4 4 len5 5;
len3 = '05AUG2004'd+1;
len4 = '05AUG2004'd+1;
len5 = '05AUG2004'd+1;
FORMAT len3 len4 len5 mmddyy10.;
RUN;

PROC PRINT DATA=date_length;
RUN;
```

Here is the resulting output. Notice that the date in len3 is different from the one in the other two variables. This is what can happen when you shrink the size of the variable too much. Instead of August 6, 2004, the value is wrong.

len3	len4	len5
08/05/2004	08/06/2004	08/06/2004