Paper #TU11

Avoiding Blind Dates with SAS®

Derek Morgan, Washington University School of Medicine

ABSTRACT

Everybody is familiar with the concept of blind dates. When the date is made, you have visions of what he or she will look like, but the reality doesn't always match that. You can have blind dates in the SAS System, too. This paper will show you how to avoid some of the possible pitfalls with dates (and times and datetimes) in your SAS code. We'll show how SAS handles dates and times through these examples of what can go wrong, and close with a discussion of Excel conversions.

WHAT'S THE FIRST THING I NEED TO KNOW?

The first thing is that SAS stores dates, times and datetimes as numbers. Dates are counted in days from a zero point of January 1, 1960. Times are counted in seconds from a zero point of midnight of the current day, and datetimes are counted in seconds since midnight, January 1, 1960. Each day that passes increments the day counter by 1, and each second that passes increments the time and datetime counters by 1. This makes it easy to calculate durations in days and seconds. Unfortunately, most references to dates and times do not use the lowest common denominator of days and seconds, respectively, and they certainly don't use January 1, 1960 for the central reference for dates. That's where the first problem comes up: how to get SAS to speak about dates and times the way we do. How do you tell SAS that the date is January 14, 1967?

```
date = “January 14, 1967”;
```

That won't get you very far. Depending on the context, you'll either get an error message telling you that you tried to put characters into a numeric value, or you'll get a character variable with the words, “January 14, 1967” stored in it. It may look OK, but if you try to do a calculation using that character variable, you'll get the SAS equivalent of a blind date.

```
DATA _NULL_;
date1 = "January 14, 1967";
date2 = "October 23, 2006";
days_in_between = date2 - date1;
PUT days_in_between = ;
RUN;
NOTE: Character values have been converted to numeric values at the places given
by:       (Line):(Column). 19   4:27
NOTE: Invalid numeric data, date2='October 23, 2006' , at line 4 column 19.
NOTE: Invalid numeric data, date1='January 14, 1967' , at line 4 column 27.

days_in_between=  .  ←——  The SAS equivalent of a blind date…
```

In order to tell SAS about a specific date, you use a “date literal.” The date literals for the two dates above are “14JAN1967”**d** and “23OCT2006”**d**. The letter “d” (bolded for emphasis only) at the end tells SAS that this is a date and not just a string of characters. When we change the date strings in the above code to date literals, we get our value for days_in_between:

```
DATA _NULL_;
date1 = "14jan1967"d;
date2 = "23oct2006"d;
days_in_between = date2 - date1;
PUT days_in_between = ;
RUN;
days_in_between=14527
```

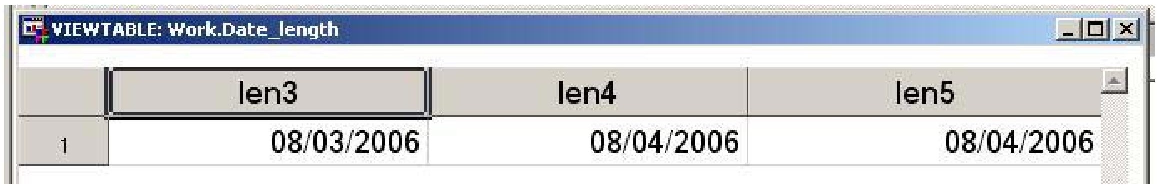
No part of the date literal is case-sensitive, that is, you can use all capital letters, all lower-case, or mixed case for the date inside the quotes and the ‘d’ can be upper or lower-case. You may use single or double quotes to enclose the literal string, but if you use double quotes, you're going to be subject to macro variable resolution. Time literals have the form “05:00:00”**t**, with a “t” instead of the “d” at the end of the character string, and datetime literals are expressed as “23oct2006:05:00:00”**dt**.

SAVING SPACE BY SHRINKING VARIABLE LENGTHS

While SAS has a default length for its numeric variables of 8, you can save space by defining smaller lengths for dates, times, and datetimes. Dates can be stored in a length of 4. Times can be stored in a length of 4, unless you need decimal fractions of seconds, then use 8 for maximum precision. Datetimes can safely be stored in a length of 6, unless you need decimal fractions of seconds, in which case, you would again use 8. For techies, if your operating system doesn’t handle half-words well, use 8 for datetimes. Why can’t you go any lower? If today’s date is August 2, 2006, and we run the following code, you’ll see.

```
DATA date_length;
LENGTH len3 3 len4 4 len5 5;
len3 = today() + 2;
len4 = len3;
len5 = len3;
FORMAT len3 len4 len5 mmddyy10.;
RUN;
```

Now let’s look at our data set:



VIEWTABLE: Work.Date_length

	len3	len4	len5
1	08/03/2006	08/04/2006	08/04/2006

While it isn’t the missing value dot, the value of len3 isn’t correct. When the numeric date value was written to the dataset, some precision was lost. (if you want to check that this happens when the data are written, place a PUT statement into the DATA step and you’ll see that the value is correct there.) This is a hit-or-miss proposition; sometimes it happens and sometimes it doesn’t, depending on the binary representation of the SAS value.

OLD DATES

SAS can go all the way back to January 1, 1582, so you’ll probably be able to work with your old dates. However, old dates can produce blind dates in SAS. You may not get a missing value, but you won’t necessarily get what you expect. Make sure that you check your century. The YEARCUTOFF option gives you the capability to define a 100-year range for two-digit year values. The default value for the YEARCUTOFF option is 1920, giving you a range of 1920-2019. Let’s demonstrate with date literals using this program:

```
OPTIONS YEARCUTOFF=1920;  /* SAS System default */
DATA yearcutoff1;
yearcutoff = "SAS System Default: 1920";
date1 = "08AUG06"d;
date2 = "15JUN48"d;
date3 = "04jan69"d;
date4 = "22oct95"d;
RUN;
OPTIONS YEARCUTOFF=1840;
DATA yearcutoff2;
yearcutoff = "1840";
date1 = "08AUG06"d;
date2 = "15JUN48"d;
date3 = "04jan69"d;
```

```
date4 = "22oct95"d;
RUN;
```

That gives us the following result::

OPTIONS YEARCUTOFF value	date1	date2	date3	date4
SAS System Default: 1920	08/08/2006	06/15/1948	01/04/1969	10/22/1995
1840	08/08/1906	06/15/1848	01/04/1869	10/22/1895

By changing the the start of the hundred year period with the YEARCUTOFF option, we've changed all of the dates without changing the date literals themselves. Any two-digit year that the SAS System has to translate, whether it's from a date literal as in the example, an ASCII file being processed with the INPUT statement and an informat, or even the INPUT() function and an informat will be affected by this option. The lesson here is to check your dates before and after processing.

FORMATS AND THE BLIND DATE

Since SAS keeps track of dates and times (and datetimes) as numbers relative to some fixed point in time, how do we get SAS to show us its dates in ways that we understand, and how can we communicate our dates to SAS? Formats are the way that SAS can translate what it understands into something that we can understand, while informats do the reverse. So how can this translation make blind dates?

First, you need to make sure that you are using the correct format or informat for your data, and the type of data you are working with. Don't try to use a date format to print out a datetime value, or use a time format to print out a date. SAS stores dates, times, and datetimes as numbers, but it does not store any context information with it. So if it isn't clear what the value represents to you, SAS won't be much help directly. (You can make an educated guess based on the maximum values and ranges of the variables involved, but it isn't foolproof.) Here's a little program to illustrate:

```
DATA _NULL_;
date = "08AUG2003"d;
time = "13:43"t;
datetime = "25JAN2005:15:52:07"dt;
PUT
 "MMDDYY10. representation of date=" date mmddyy10. /
 "MONYY7. representation of date=" date monyy7. /
 "TIMEAMPM9. representation of date=" date timeampm9. /
 "DTMONYY7. representation of date=" date dtmonyy. /
 "When value of date is used as a SAS *datetime* value, the date represented is:"
  date datetime20. /
 "DATETIME20. representation of time=" time datetime20. /
 "DTMONYY7. representation of time=" time dtmonyy7. /
 "TIMEAMPM9. representation of time=" time timeampm9. /
 "MONYY7. representation of time=" time monyy7. /
 "When value of time is used as a SAS *date* value, the date represented is:"
 time mmddyy10. /
 "DATETIME20. representation of datetime=" datetime datetime20. /
 "DTMONYY7. representation of datetime=" datetime dtmonyy7. /
 "TIMEAMPM9. representation of datetime=" date timeampm9. /
 "MONYY7. representation of datetime=" datetime monyy7. /
 "When value of datetime is used as a SAS *date* value, the date represented is:"
 datetime mmddyy10.;
RUN;
```

To make it a little easier to compare and contrast, here are the results in tabular form. Any DATA step and PROC REPORT or PROC TABULATE manipulations necessary to produce this output are left as an exercise for the reader.

		Date Formats		Time Format	Datetime Formats	
Variable	**Value in SAS**	**Using MMDDYY10. format**	**Using MONYY. format**	**Using TIMEAMPM9. format**	**Using DTMONYY7. format**	**Using DATETIME20. format**
Date	15925	08/08/2003	AUG2003	4:25:25 AM	JAN1960	01JAN1960:04:25:25
Time	49380	03/13/2095	MAR2095	1:43:00 PM	JAN1960	01JAN1960:13:43:00
Datetime	1422287527	**********	*******	***3:52:07 PM***	JAN2005	25JAN2005:15:52:07

The first thing that you notice is that the datetime value gives you a bunch of asterisks when you try to format it as a date. The date is so far in the future that it can’t be represented with a four-digit year, but that's the only blatant indication that something’s not quite right. Why the discrepancy on the others? When you try to translate a date value with a time format, you are translating days since 1/1/1960 with something that’s supposed to translate seconds since midnight. 15,925 seconds after midnight is 4:25:25 in the morning. If you translate 15,925 as seconds after midnight, 1/1/1960, which is the datetime convention, you get 4:25:25 AM on January 1, 1960. Similarly, if you translate 49,380 as days since 1/1/1960, you get March 13, 2095. Finally, note the cell in italics. There’s absolutely nothing to indicate that something is wrong here. Why do we get a normal-looking time? The TIMEAMPM format gives times from 12:00 AM to 11:59 PM, so any value greater than 86400 (the number of seconds in a day) just cycles into the next day. Therefore, you are getting the result of MOD(value,86400).

NEED A FORMAT FOR YOUR DATE?

Although there are many formats that are built into SAS, you may find yourself with a case where you can’t find a format that displays your date, time, or datetime the way that you want. Don’t panic. You can use a custom format. You can create a format to show off your dates. There are two ways to do this, and the first is with the VALUE statement in PROC FORMAT. You define a range for the values using date, time, or datetime constants, and you can tell SAS what to print out instead of the date. Here’s a sample program that will create a format to display whether a contract is scheduled for arbitration or renegotiation based on the expiration date of the contract:

```
PROC FORMAT LIBRARY=library;
VALUE contrct
LOW-‘31dec2005’d="INVALID"
‘01JAN2006’D-’31JUL2006’d= "ARBITRATION"
‘01AUG2006’d - ‘31DEC2006’d = "RENEGOTIATION"
‘01JAN2007’d - high=[MONYY7.];   /* INSTRUCTS SAS TO USE THE
                                  MONYY7. FORMAT FOR VALUES BEYOND 2006*/
RUN;

PROC PRINT DATA= contracts;
ID contract_num;
VAR exp_date exp_date_raw;
FORMAT exp_date contrct. exp_date_raw MMDDYY10.;
RUN;
```

Here's some of the output – instead of the date, our format classifies the date values and translates them into categorical text.

contract_num	exp_date	exp_date_raw
5829014	INVALID	12/06/2005
9330471	RENEGOTIATION	09/21/2006
6051271	APR2007	04/11/2007
2301911	ARBITRATION	01/23/2006
6894300	RENEGOTIATION	08/21/2006

So where can you go wrong here? Lot's of places, actually. Let's examine the code for our format:

```
1    PROC FORMAT LIBRARY=LIBRARY;
2    VALUE CONTRCT
3    LOW-'31dec2005'd="INVALID"
4    '01JAN2006'D-'31JUL2006'd= "ARBITRATION"
5    '01AUG2006'd - '31DEC2006'd = "RENEGOTIATION"
6    '01JAN2007'd - HIGH=[monyy7.];   /* Instructs SAS to use the
                                        MONYY7. format for values beyond 2006*/
7    RUN;
```

First, if you forget the "D" to indicate that the value is a date constant, you're going to get an error from lines 3-6. Notice that line 3 uses the special value "LOW". Without it, any date before January 1, 2006 will display as the actual SAS numeric value. Similarly, line 6 accounts for values in the future by using the special value "HIGH". However, instead of setting it to print categorical text, we've told SAS to use one of its own date formats if the date is after December 31, 2006. That's why there's a format name enclosed in brackets after the equals sign. Without the format name, there's no formatting associated with the SAS date value.

PRETTY AS A PICTURE

The second way to create your own format for your date, time, or datetime is with a picture format. Picture formats allow you to create a representation of your data by describing what you want it to look like. There are special formatting directives to allow you to represent dates, times and datetime values. These directives are **case-sensitive**. You will also need to use the DATATYPE= option in your PICTURE statement. DATATYPE is DATE, TIME, or DATETIME, and it indicates the type of value you are formatting.

Here are the directives:

%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. Put a zero between the percent sign and the “d” to have a leading zero in the display.
%H	Hour (24-hour clock) as a decimal number (0-23), with no leading zero. Put a zero between the percent sign and the “H” to have a leading zero in the display.
%I	Hour (12-hour clock) as a decimal number (1-12), with no leading zero. Put a zero between the percent sign and the “I” to have a leading zero in the display.
%j	Day of the year as a decimal number (1-366), with no leading zero. Put a zero between the percent sign and the “j” to have a leading zero in the display.
%m	Month as a decimal number (1-12), with no leading zero. Put a zero between the percent sign and the “m” to have a leading zero in the display.

%M	Minute as a decimal number (0-59), with no leading zero. Put a zero between the percent sign and the “M” to have a leading zero in the display.
%p	Either AM or PM.
%S	Second as a decimal number (0-59), with no leading zero. Put a zero between the percent sign and the “S” to have a leading zero in the display.
%U	Week number of the year (Sunday as the first day of the week) as a decimal number (0-53), with no leading zero. Put a zero between the percent sign and the “U” to have a leading zero in the display.
%w	Weekday as a decimal number, where 1 is Sunday, and Saturday is 7.
%y	Year without century as a decimal number (0-99), with no leading zero. Put a zero between the percent sign and the “y” to have a leading zero in the display.
%Y	Year with century as a decimal number (four-digit year).
%%	The percent character (%).

Here's a simple example of using the date directives to create an enhanced date display with the day of the year.

```
PROC FORMAT;
PICTURE xdate
. - .z = "No Date Given"
LOW - HIGH = '%B %d, %Y is day %j of %Y' (DATATYPE=DATE);
RUN;
PROC PRINT DATA=pictest;
VAR date;
FORMAT date xdate40.;
RUN;
```

Let's look at the output for several pseudo-random dates:

date
No Date Given
August 3, 2005 is day 215 of 2005
November 27, 2005 is day 331 of 2005
January 14, 2008 is day 14 of 2008
October 6, 2007 is day 279 of 2007
February 13, 2006 is day 44 of 2006
May 18, 2007 is day 138 of 2007
August 12, 2007 is day 224 of 2007
October 2, 2005 is day 275 of 2005
March 22, 2007 is day 81 of 2007

What can lead you to a blind date with this? First, make sure that you have defined the correct DATATYPE. Otherwise, you're not going to get a correct representation, just like you would by using the wrong type of format with the built-in SAS formats. Second, you need to make sure that you use a format length that is long enough to show all of your text. The default length of a picture format is the number of characters between the quotes in the picture. However, there may be more characters in your output. That's why the format length in the PROC PRINT statement above is set to 40 ("xdate40."). If we were to allow the PROC PRINT to use the format with its default length, this would be the output: we would get.

date
.
August 3, 2005 is day 215
November 27, 2005 is day
January 14, 2008 is day 1
October 6, 2007 is day 27
February 13, 2006 is day
May 18, 2007 is day 138 o
August 12, 2007 is day 22
October 2, 2005 is day 27
March 22, 2007 is day 81

Oops. This happens because the default length of the format is 25, which won't accommodate all of the text in the formatted value. Since each of the format directives are only two characters long, the values they display can be longer than that. For example, the “%B” format directive represents the full month name, which is always going to be longer than two characters. This is why you must give a format length when you use your custom picture formats with dates, times, and datetimes.

READING DATES AND TIMES AND DATETIMES

So far, our examples have all used date constants, but you can't put a date constant everywhere you need a date, such as in data. If you are converting data from a flat file, then you will need to use informats to read the data. To read date, time, or datetime data from a flat file, you will need both the formatted INPUT statement and an informat. Here's an example of a flat file with dates:

```
10/26/2000
09/03/1998
05/14/1967
08/25/1989
07/01/2004
03/16/2001
03/16/1971
04/03/1968
09/25/1965
```

To read this file, you would use this DATA step:

```
DATA read_dates;
INFILE "c:\book\examples\a_few_dates.txt";
INPUT @1 sample_date MMDDYY10.;
RUN;
```

And here's a log of a program to display what is stored.

```
DATA _NULL_;
SET read_dates;
PUT "Sample date without formatting " sample_date /
    "Sample date with WORDDAT format " sample_date WORDDATE.;
RUN;

Sample date without formatting 14909
Sample date with WORDDATE. format   October 26, 2000
Sample date without formatting 14125
Sample date with WORDDATE. format  September 3, 1998
Sample date without formatting 2690
Sample date with WORDDATE. format       May 14, 1967
Sample date without formatting 10829
Sample date with WORDDATE. format    August 25, 1989
Sample date without formatting 16253
Sample date with WORDDATE. format       July 1, 2004
Sample date without formatting 15050
Sample date with WORDDATE. format     March 16, 2001
Sample date without formatting 4092
Sample date with WORDDATE. format     March 16, 1971
Sample date without formatting 3015
Sample date with WORDDATE. format      April 3, 1968
Sample date without formatting 2094
Sample date with WORDDATE. format September 25, 1965
```

Since we looked at the file first, we knew that all of the data looked like "mm/dd/yyyy", and we simply told the INPUT statement that was what it would see when it read the field. By specifying that informat, we told SAS how to translate what seems to be a character string ("/" is not a number) into a SAS date value. Blind dates are easy to come by here: if you use the wrong informat for your data, things will definitely go wrong. In most cases, using the wrong informat will give you an error, but you need to be careful with some of the informats that differ only in the order of month, day, and/or year. The MMDDYY. informat will not give you the same result as the DDMMYY. informat, and it wouldn't give you any message that anything was abnormal until the middle two digits in the data field were greater than twelve.

READING DATES AND TIMES AND DATETIMES THE EASY WAY

Version 9 addressed one problem with reading dates from flat files into SAS: you had to know what your date and time data looked like before you could process it from a flat file into SAS. The ANYDTDTE., ANYDTDTM., and ANYDTTME. informats will translate dates, times, and datetime values respectively, into their corresponding SAS System values without having to know the representation of these dates, datetime, and time values in advance. Pretty nifty. Of course, there are a couple of warnings: if you are going to leave it to SAS to figure out what the data look like, it will add some overhead to the processing of your flat file. How much extra overhead is involved? Let's read 10,000 date records with SAS for Windows XP on a PC and find out. Here's a partial log:

```
241  DATA test;
242  INFILE "examples/datefile.txt" PAD MISSOVER;
243  INPUT @1 date mmddyy10.;
244  RUN;

NOTE: 10000 records were read from the infile "examples/datefile.txt".
      The minimum record length was 10.
      The maximum record length was 10.
NOTE: The data set WORK.TEST has 10000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds

246  DATA test2;
247  INFILE "examples/datefile.txt" PAD MISSOVER;
248  INPUT @1 date anydtdte10.;
249  RUN;

NOTE: 10000 records were read from the infile "examples/datefile.txt".
      The minimum record length was 10.
      The maximum record length was 10.
NOTE: The data set WORK.TEST2 has 10000 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds
```

Using the ANYDTDTE. format takes four times as long in CPU time to read this relatively small flat file than using the MMDDYY. format does. So what if it takes more time? Doesn't it seem to solve all of your problems? Where can you get blind dates from this? Well, what if you have a date such as 06/11/2005, or worse, 06/11/05? Is it June 11, 2005, November 6, 2005, or November 5, 2006? There's an option that will allow you to tell SAS what it should be. The DATESTYLE option comes in seven flavors:

MDY	Sets the default order as month, day, year. 06-11-05" would be translated as June 11, 2006	**YDM**	Sets the default order as year, day, month. "06-11-05" would be translated as May 11, 2006
MYD	Sets the default order as month, year, day. "06-11-05 would be translated as June 5, 2011	**DMY**	Sets the default order as day, month, year. "06-11-05" would be translated as November 6, 2005
YMD	Sets the default order as year, month, day. "06-11-05" would be translated as November 5, 2006	**DYM**	Sets the default order as day, year, month. "06-11-05" would be translated as May 6, 2011
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, "06-11-05" would be translated as June 11, 2005		

Although it may seem just that easy to resolve ambiguous dates, it's not foolproof.

AM I READING ENOUGH?

One more issue with reading dates and times from a flat file is making sure that you are reading the correct number of characters in the field. All of the informats have a length specification, which is how many characters are to be processed using the informat. For example, the MMDDYY10. informat reads ten characters, and will attempt to translate it as mm-dd-yyyy, although the exact separator between month, day, and year may vary. If you forget the length specification, each informat has its own default length. This default may not match your data. The bottom line is that when you are reading in date values from a flat file, it's best to look at the data that you've produced from the flat file before you actually use it.

CREATING DATES, TIMES AND DATETIMES FROM NUMBERS

There is a whole group of functions in SAS that are dedicated to creating the date, time, and datetime values from the individual elements such as month, year, hour, and minute. The MDY() function takes individual values for month, day and year, and converts them to a SAS date value. Similarly, the HMS() function creates SAS time values from hours, minutes, and seconds. Finally, the DHMS() function yields SAS datetime values from dates, hours, minutes, and seconds. Each of the functions needs all of its component arguments, e.g., the DHMS() function needs non-missing values for date, hour, minute, and second. If you are missing any one of these components, you will get a missing value as the result.

MAKING DATES

Many methods are available for filling in missing values for month and day before conversion, but the "best guess" method is one that we often use.

```
IF year LT 0 THEN
   date = .E;
ELSE DO;
   IF month LT 1 AND day LT 1 THEN DO;
      month = 6;
      day = 30;
   END;
   ELSE DO;
      IF day LT 1 THEN DO;
         SELECT(month);
           WHEN(2) day=14;
           OTHERWISE day = 15;
         END;
      END:
   END;
   date = MDY(month,day,year);
END;
```

The net effect of this code is that any missing dates due to missing year values will stick out like a sore thumb, while if we have a year, but no date, the date is set to approximately mid-year (June 30). If only the day is missing, we approximate the middle of the month (14 for February, 15 for all other months).

EXTRACTING MONTH, DAY, YEAR, ETC. FROM YOUR NON-BLIND DATE

There are also functions that will pull the individual component values from SAS date, time and datetime values. The main thing you need to be careful of is that you use the appropriate function for the type of value you are using. Don't try to use the DAY() function to extract a day from your datetime value. It won't work right, as the next example shows.

```
DATA _NULL_;
datetime = "27JAN1960:12:35:00"dt;
time = "12:35:00"t;
date = "27JAN1960"d;
day_datim = DAY(datetime);
day_tm = DAY(time);
day_date = DAY(date);
PUT datetime= +3 time= +3 date= ;
PUT day_datim= +3 day_tm= +3 day_date=;
RUN;
datetime=2291700    time=45300    date=26
day_datim=20 day_tm=10 day_date=27
```

The first line of output shows the actual values that SAS is operating on. In the above example, I purposely used an early datetime value (2,291,700 seconds relative to midnight, January 1, 1960,) because it gives me a non-missing value, albeit incorrect, for the day. If you use a datetime sometime in March of 1960, SAS will give you an error message. However, it doesn't give an error message because it knows that it's a datetime value; it's trying to figure out a day for a value that's too large for the function to work. In that vein, here's a trickier one:

```
DATA _NULL_;
datetime = "27MAR1998:12:15:00"dt;
time = "12:15:00"t;
date = "27MAR1998"d;
minit_datim = MINUTE(datetime);
minit_tm = MINUTE(time);
minit_date = MINUTE(date);
PUT datetime= +3 time= +3 date= ;
PUT minit_datim= +3 minit_tm= +3 minit_date=;
RUN;
datetime=1206620100    time=44100 date=13965
minit_datim=15    minit_tm=15    minit_date=52
```

What happened here? It worked just fine for the datetime value, didn't it? Not exactly. Remember that both times and datetimes are stored in seconds since midnight, and that times cycle every 86,400 seconds. Any value greater than 86400 (the number of seconds in a day) just cycles into the next day. Therefore, you are getting the result of MOD(value,86400). So, while the time extraction functions seem to work with datetimes, it's a bad habit to get into. You can also clearly see that trying to get the minute out of the date value gives you a truly blind date.

One last example of a date function that can give you blind dates is the JULDATE() function. It will convert a date value into a numeric representation of a Julian date. The output it yields is affected by the YEARCUTOFF option. It will return a four- or five-digit value if the year portion of the date falls within the 100-year span defined by the YEARCUTOFF= option, regardless of whether you're using an argument with a two- or four- digit year, or a SAS date value. Since it returns a numeric value, leading zeroes are not significant, and are dropped. If that "200", or that "19" are going to be important to you, use the Y2Kcompliant JULDATE7() function, which will always return a seven-digit number.

```
DATA _NULL_;
juldate1 = JULDATE("27JAN1990"d);
juldate2 = JULDATE("27JAN1890"d);
juldate7_1 = JULDATE7("27JAN1990"d);
juldate7_2 = JULDATE7("27JAN1890"d);
RUN;
```

Here's the result in a table:

Date	Using JULDATE()	Using JULDATE7()	What happened?
January 27, 1990	*90027*	*1990027*	They're different due to the YEARCUTOFF processing of the argument.
January 27, 1890	1890027	1890027	Same result, because the date argument to the function is outside of the YEARCUTOFF range (19202019)

INTERVALS

Intervals are prone to giving people blind dates. They are very powerful, but frequently misunderstood, especially when they are used in association with the two interval calculation functions INTCK() and INTNX(). The INTCK() function counts the number of intervals between two given dates, times, or datetimes, while the INTNX() function takes a given date, time or datetime and increments it by a given number of intervals. SAS has several standard intervals already defined. These intervals represent periods of time such as days, weeks, months, and quarters, to name a few. A complete list of the SAS intervals, along with the default starting points for each type of interval is on the following page.

Interval Name	Definition	Default Starting Point
DAY	Daily intervals	Each day
WEEK	Weekly intervals of seven days	Each Sunday
WEEKDAYdaysW	Daily intervals with Friday-Saturday-Sunday counted as the same day (five-day work week with a Saturday-Sunday weekend). days identifies the individual numbers of the weekend day(s) by number (1=Sunday ... 7=Saturday). By default, days="17", so the default interval is WEEKDAY17W.	Each day
TENDAY	Ten-day intervals (a U.S. automobile industry convention)	1st, 11th, and 21st of each month
SEMIMONTH	Half-month intervals	First and sixteenth of each month
MONTH	Monthly intervals	First of each month
QTR	Quarterly (three-month) intervals	1-Jan 1-Apr 1-Jul 1-Oct
SEMIYEAR	Semi-annual (six-month) intervals	1- Jan 1 Jul
YEAR	Yearly intervals	1-Jan
DTDAY	Daily intervals used with datetime values	Each day
DTWEEK	Weekly intervals of seven days used with datetime values	Each Sunday
DTWEEKDAYdaysW	Daily intervals with Friday-Saturday-Sunday counted as the same day (five-day work week with a Saturday-Sunday weekend). days identifies the individual weekend days by number (1=Sunday ... 7=Saturday). By default, days="17", so the default interval is DTWEEKDAY17W. This interval is only used with datetime values.	Each day
DTTENDAY	Ten-day intervals (a U.S. automobile industry convention) used with datetime values	1st, 11th, and 21st of each month
DTSEMIMONTH	Half-month intervals used with datetime values	First and sixteenth of each month
DTMONTH	Monthly intervals used with datetime values	First of each month
DTQTR	Quarterly (three-month) intervals used with datetime values	1-Jan 1-Apr 1-Jul 1-Oct
DTSEMIYEAR	Semiannual (six-month) intervals used with datetime values	1- Jan 1 Jul
DTYEAR	Yearly intervals used with datetime values	1-Jan
DTSECOND	Second intervals used with datetime values	Seconds
DTMINUTE	Minute intervals used with datetime values	Minutes
DTHOUR	Hour intervals used with datetime values	Hours
SECOND	Second intervals used with datetime values	Seconds
MINUTE	Minute intervals used with datetime values	Minutes
HOUR	Hourly intervals used with datetime values	Hours

These may seem quite self-explanatory, but if you just jump right in and use these intervals without reading about them, you will end up with many blind dates.

WHEN IS A YEAR NOT A YEAR?

The syntax for the INTCK() function is: **INTCK(interval,start-of-period,end-of-period);** To illustrate the blind date potential, consider the following program:

```
DATA _NULL_;
v1 = INTCK('YEAR','01jan2005'd,'01jan2006'd);
v2 = INTCK('YEAR','31dec2005'd,'01jan2006'd);
PUT v1= +3 v2= +3;
RUN;
v1=1 v2=1
```

Now wait a minute. I know that a year from December 31, 2005 isn't January 1, 2006. What happened? SAS intervals aren't a shortcut to doing the math. The INTCK() function counts the number of times that the period ***interval*** begins between start-of-period and end-of-period. It does not count the number of complete intervals between start-of-period and end-of-period. Therefore, for any date in 2005, the first year period starts on January 1, 2006, which means that for any date in 2005, the number of YEAR intervals INTCK will return is 1 for any given date in 2006. There's big potential for blind dates here if you misunderstand how INTCK() goes about its business.

The big thing to remember about intervals is that they are based on the beginning point of the interval, and not the beginning date given to the function.

BUT I DON'T WANT MY YEAR TO START ON JANUARY 1

Since the default starting point of an interval is at the beginning of the interval, it would seem that SAS has a blind spot when it comes to figuring out intervals that do not coincide with that. There is a way to shift the starting point of any given interval by creating your own interval definition. For example, what if you wanted to know the number of YEAR intervals between two dates, but instead of calculating calendar years, you wanted to calculate your company's fiscal year, which starts on February 1? You tell SAS how many periods to shift. Each interval has a shift unit. For years, the shift unit is months, so you will tell SAS to shift the starting point of the year in terms of months. A shifted interval is the interval name, followed by a period, and the number of periods to shift. To shift the start of the YEAR interval to February 1, you would use the interval "YEAR.2".

Here's where the blind date comes in: when you count the number of periods to shift, you need to include the beginning of the period. We are not shifting the start of the year interval by one month to move to February; we're moving the start of the year to the second month. If you don't account for this, you'll be off by one unit. Another handy way to think of it is that the YEAR.1 interval is the same as the YEAR interval.

EXCEL

Let's not overlook the potential for blind dates whenever you convert from Microsoft Excel into SAS or vice versa. The worst offender? Old dates. Microsoft Excel uses a zero point of January 1, 1900. It cannot count backwards from zero. Therefore, any date prior to 1900 is represented as text, not as an Excel date value. What do you if you have dates before January 1, 1900 to convert? Going to Excel, you will have to store them as character strings, and you won't be able to use any of the Excel math functions on them. On the other hand, if you are going from Excel to SAS, then you can use an INPUT statement and a DATA step to process a CSV file. The IMPORT wizard and/or procedure? They'll give you the dreaded dot.

SUMMARY

The way that SAS handles dates, times, and datetimes allows for a great deal of flexibility in the display, manipulation, and computation of these important pieces of data. However, that flexibility can also make it easy to do things wrong. A solid basic understanding of dates and times in SAS is necessary to use this powerful tool correctly. Dates, times, and datetimes are stored as numbers in the SAS system. Dates are counted in days from a zero point of January 1, 1960. Times are counted in seconds from a zero point of midnight of the current day, and datetimes are counted in seconds since midnight, January 1, 1960. Each day that passes increments the day counter by 1, and each second that passes increments the time and datetime counters by 1.

The display of SAS dates and times are accomplished by using formats. There are many built-in formats in SAS. Each of these displays a date, time, or a datetime value as something we can recognize. Date formats are specific to date values, time formats are specific to time values, and datetime formats are specific to datetime values. If you use the wrong type of format, your display will be incorrect. If none of the built-in formats meet your needs, you can create your own, either by defining the display to correlate with certain values, or by defining the display with a pictorial template using symbolic directives.

In order to translate dates and times as we know them into the values recognized by SAS, we use informats. As with formats, there are many built-in informats, designed to translate standard alphanumeric or numeric representations of dates into SAS date, time, or datetime values. You need to specify the number of characters to be processed by the informat, and you should use the correct informat for the alphanumeric characters you are processing. If you don't know, you can use the ANYDATE sereis of informats, but be prepared for a performance hit.

You may extract the individual components of a SAS date, time, or datetime value, such as month number, day, or hour as examples. However, you need to make sure that you do not try to extract a month from a time, or an hour from a date. You may also create a SAS date, time, or datetime value from individual components. Whenever you do this, remember that all of the arguments to the MDY(), HMS(), and DHMS() functions must be present for them to work.

It is frequently useful to refer to periods of time by familiar reference, such as year or quarter. SAS handles this with intervals. There are several such standard periods defined in SAS, and two functions that use them, INTCK() and INTNX(). The most important thing to remember is that these functions use SAS intervals by measuring the intervals from the start point of the interval, not the dates supplied to the functions as arguments.

There are many more uses of dates, many more functions relating to dates and times and datetimes, and more capabilities of intervals than have been shown in this brief paper. However, the flexibility of SAS and the tools it provides for manipulation of dates, times, and datetimes should allow you to accomplish whatever you need to.

REFERENCES

1. Morgan, Derek P. 2006. *The Essential Guide to SAS® Dates and Times*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION:

Further inquiries are welcome to:
Derek Morgan
Division of Statistical Genomics
Washington University Medical School
Box 8506, 4444 Forest Park Blvd.
St. Louis, MO 63108
Phone: (314) 362-3685 FAX: (314) 362-4227
E-mail: derek@wustl.edu