MWSUG 2017 – Paper SA03

The Essentials of SAS® Dates and Times

Derek Morgan, PAREXEL International, Billerica, MA

ABSTRACT

The first thing you need to know is that SAS® software stores dates and times as numbers. However, this is not the only thing that you need to know, and this presentation will give you a solid base for working with dates and times in SAS. It will also introduce you to functions and features that will enable you to manipulate your dates and times with surprising flexibility. This paper will also show you some of the possible pitfalls with dates (and times and datetimes) in your SAS code, and how to avoid them. We'll show you how the SAS System handles dates and times through examples, including the ISO 8601 formats and informats, how to use dates and times in TITLE and/or FOOTNOTE statements, and close with a brief discussion of Excel conversions.

WHAT'S THE FIRST THING I NEED TO KNOW?

The first thing is, of course, 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 one, and each second that passes increments the time and datetime counters by one. 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, and midnight as their central references. That is 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 will not get you very far. Depending on the context, you will get an error message telling you that you tried to put characters into a numeric value, or you will get a character variable with the words, "January 14, 1967" stored in it. It may look okay, but if you try to do a calculation using that character variable, you will get a missing value.

```
DATA _NULL_;
date1 = "January 14, 1967";
date2 = "September 4, 2014";
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='September 4, 2014' , at line 4
column 19.
NOTE: Invalid numeric data, date1='January 14, 1967' , at line 4 column
27.
days_in_between = •    ← (the dreaded missing value dot)
```

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 "04SEP2014"d. The letter "d" at the end tells SAS that this is a date, not a string of characters, so the code becomes:

```
DATA _NULL_;
date1 = "14jan1967"d;
date2 = "04sep2014"d;
days_in_between = date2 - date1;
PUT days_in_between = ;
RUN;

days_in_between=17400
```

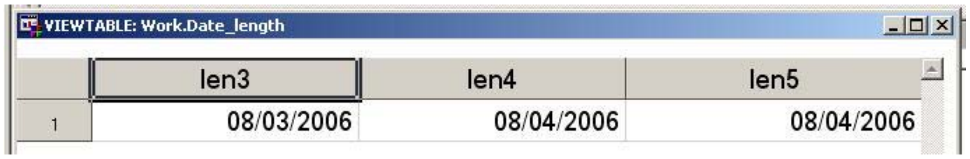
No part of the date literal is case-sensitive, that is, you can use all capital letters, all lower-case, or mixed case for the date inside the quotes and the 'd' can be upper or lower-case. You may use single or double quotes to enclose the literal string, but if you use double quotes, you will be subject to macro variable resolution, which means that an ampersand (&) may cause unexpected results. Time and datetime literals are expressed in a similar fashion; however, instead of the letter "d", they are followed by the letter "t" for time, or the letters "dt" for datetimes. Time literals are expressed with a twenty-four hour clock in the form "05:00:00"t, and datetime literals are expressed as "04sep2017:05:00:00"dt.

SAVING SPACE BY SHRINKING VARIABLE LENGTHS

While SAS has a default length of eight for numeric variables, you can save space by defining smaller lengths for dates, times, and datetimes. Dates can be stored in a length of four. Times can be stored in a length of four, unless you need decimal fractions of seconds; then you would use eight for maximum precision. Datetimes can safely be stored in a length of six, unless you need decimal fractions of seconds, in which case you would again use eight. For techies, if your operating system doesn't handle half-words, use eight for datetimes. Why can't you go any lower? Given a date of August 4, 2006, if we run the following code, you will see.

```
DATA date_length;
LENGTH len3 3 len4 4 len5 5;
len3 = "04AUG2006"d + 2;
len4 = len3;
len5 = len3;
FORMAT len3 len4 len5 mmddyy10.;
RUN;
```

Now let's look at our data set:



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

While it isn't the missing value dot, you can see that the value of len3 is not correct. When the numeric date value was written to the dataset, some precision was lost. This is a hit-or-miss proposition; sometimes it happens and sometimes it doesn't. Do not take the risk.

HISTORICAL DATES

SAS can go all the way back to January 1, 1582, so you will likely be able to work with historical dates. However, historical dates have the potential to produce incorrect values in SAS. You may not get a missing value, but 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 in version 9.4 of SAS is 1926, giving you a range of 1926-2025. Let's demonstrate with date literals using the following code, and then put the two datasets together in Result 1:

```
OPTIONS YEARCUTOFF=1926;  /* SAS System default */
DATA yearcutoff1;
yearcutoff = "SAS System Default: 1926";
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;
```

OPTIONS YEARCUTOFF value		date1		date2		date3		date4
1926	08AUG06	08/08/2006	15JUN48	06/15/1948	04JAN69	01/04/1969	22OCT95	10/22/1995
1840		08/08/1906		06/15/1848		01/04/1869		10/22/1895

Result 1: Effects of the YEARCUTOFF option on identical dates.

As you can see, identical date literals can give you completely different results based on the value of this option. Any two-digit year that SAS has to translate, whether it is from a date literal as shown in the above 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. However, dates that are already stored as SAS dates are NOT affected by this option. SAS dates are simply stored as the number of days from January 1, 1960, and so the two-digit vs. four-digit year doesn't matter. The lesson here is to check your dates before and after processing.

FORMATS AND TRANSLATING SAS DATES

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 built-in translation fail?

First, you need to make sure you are using the correct format or informat for your data, and the type of data you are working with. Do not 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. Unfortunately, this means that if it is not clear what the value represents to you, SAS will not be of much help directly. (You can make an educated guess based on the maximum values and ranges of the variables involved, but that method is not foolproof, and it would be data-dependent.) To illustrate, let's take a date value representing August 8, 2014, a time value of 11:43 AM, and a datetime value of 3:52:07 PM on January 25, 2015, and display them with a range of formats in table Result 2.

		Date Formats		Time Format	Datetime Formats	
Intended Use for Variable	**Value in SAS**	**Using MMDDYY10. format**	**Using MONYY7. format**	**Using TIMEAMPM11. format**	**Using DTMONYY7. format**	**Using DATETIME19. format**
Date	19943	08/08/2014	AUG2014	5:32:23 AM	JAN1960	01JAN1960:05:32:23
Time	42180	06/26/2075	JUN2075	11:43:00 AM	JAN1960	01JAN1960:11:43:00
Datetime	1737820327	**********	*******	***3:52:07 PM***	JAN2015	25JAN2015:15:52:07

Result 2. The importance of context when using formats to represent SAS date and time values.

The first thing you should notice is that the datetime value gives you several asterisks when you try to format it as a date. The date represented by the value 1,737,820,327 is so far in the future that it cannot be represented by a four-digit year, but that is the only blatant indication that something's not quite right. Why is there a discrepancy on the others? When you try to translate a date value with a time format, you are translating days since January 1, 1960 using something designed to translate seconds since midnight. 19,943 seconds after midnight is 5:32:23 in the morning. If you translate 19,943 as seconds after midnight of January 1, 1960, which is the definition of a datetime, you get 5:32:23 AM on January 1, 1960. Similarly, if you translate 42,180 as days since January 1, 1960, you get June 26, 2075. Finally, note the cell in italics. There is 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 86,400 (the number of seconds in a day) just cycles into the next day. Therefore, you are getting the result of the calculation MOD(1737820327,86400), which is 57,127, and translates into a time of 3:52:07 PM using the time scale of seconds after midnight.

NEED A FORMAT FOR YOUR DATE?

Although there are many formats built into SAS, you may find yourself in a position where you cannot find a format that displays your date, time, or datetime the way you want. Don't panic. You can create and use a custom format to show off your dates. There are two ways to do this and they both require using the FORMAT procedure. The first way uses the VALUE statement. You define a range for the values using date, time, or datetime constants, and then 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;
VALUE contrct
LOW-'15nov2013'd = "EXPIRED"
'16NOV2013'd-'15nov2014'd = "RENEGOTIATION"
'16NOV2014'd - '15nov2016'd = "ARBITRATION"
'16nov2016'd - high= [MONYY7.];  /* INSTRUCTS SAS TO USE THE MONYY7. FORMAT
FOR VALUES
                                    BEYOND NOVEMBER 16, 2016 */
RUN;
```

This is a look at a few records of the raw data:

Contract Number	Expiration Date
5829014	11/06/2013
9330471	09/21/2015
6051271	04/11/2015

Here is some of the output–instead of printing the date for the variable EXP_DATE, our format classifies the date values and translates them into categorical text. We are going to reformat the display of the raw date as well. Using aliases in the COLUMNS statement below, we can turn these two columns from the dataset we create into an ordered list by date using differing formats.

```
PROC REPORT DATA=contracts NOWD;
COLUMNS exp_date_raw contract_num exp_date_raw=exp_date
exp_date_raw=exp_date_disp;
DEFINE exp_date_raw / NOPRINT ORDER;
DEFINE contract_num / "Contract Number";
DEFINE exp_date / FORMAT=contrct. "Negotiation Status at End-of-Term";
DEFINE exp_date_disp / FORMAT=worddate. "Expiration Date";
RUN;
```

Contract Number	Negotiation Status at End-of-Term	Expiration Date
5829014	EXPIRED	November 6, 2013
2301911	RENEGOTIATION	January 23, 2014
1540956	ARBITRATION	December 1, 2014
6051271	ARBITRATION	April 11, 2015
9330471	ARBITRATION	September 21, 2015
6894300	ARBITRATION	August 21, 2016
7465502	NOV2016	November 18, 2016

Result 3: Custom Format Using the VALUE Statement

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

```
1  PROC FORMAT;
2  VALUE contrct
3  LOW-'15nov2013'd="EXPIRED"
4  '16NOV2013'd-'15nov2014'd="RENEGOTIATION"
5  '16NOV2014'd - '15nov2016'd = "ARBITRATION"
6  '16nov2016'd-high=[MONYY7.];  /* INSTRUCTS SAS TO USE THE MONYY7.
   FORMAT FOR
                                  VALUES BEYOND NOVEMBER 16, 2016 */
7  RUN;
```

First, if you forget the "d" to indicate that the value is a date constant, you are going to get an error from lines 3-6. Notice that line 3 uses the special value "LOW". Without it, any date before November 16, 2013, 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 display categorical text, we have told SAS to use one of its own date formats if the date is after November 16, 2016. That is why there is a format name enclosed in brackets after the equal sign. Without the format name, there would be no formatting associated with the SAS date value, and all you would see displayed would be the number of days since January 1, 1960.

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 to indicate 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 (%).

Table 1: SAS Date Directives for use with PICTURE formats

Here is 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); ❶
RUN;
```

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

SAS Date Value	Date Formatted Using WORDDATE.	Date Formatted Using Custom Format XDATE40.	Date Formatted Using Custom Format XDATE. WITHOUT a Length Specification
.	.	No Date Given	No Date Given
19703	December 11, 2013	December 11, 2013 is day 345 of 2013	December 11, 2013 is day
19724	January 1, 2014	January 1, 2014 is day 1 of 2014	January 1, 2014 is day 1
19765	February 11, 2014	February 11, 2014 is day 42 of 2014	February 11, 2014 is day
19849	May 6, 2014	May 6, 2014 is day 126 of 2014	May 6, 2014 is day 126 of
19860	May 17, 2014	May 17, 2014 is day 137 of 2014	May 17, 2014 is day 137 o
19920	July 16, 2014	July 16, 2014 is day 197 of 2014	July 16, 2014 is day 197
20033	November 6, 2014	November 6, 2014 is day 310 of 2014	November 6, 2014 is day 3

Result 4: Example of a Custom-Designed Date Format Using Date Directives in a PICTURE statement

Well, the third column is impressive. Nothing other than the XDATE format we created was used to produce this. So where can you go wrong with this? First, remember that you cannot translate date values with time or datetime formats. Since we are working with date values here, make sure that you have defined the DATATYPE correctly (line 4, in the code above.) Otherwise, SAS will interpret the data as seconds after midnight when DATATYPE=TIME or seconds after midnight on January 1, 1960 if DATATYPE=DATETIME, and your result will be spectacularly incorrect. In essence, this is the same issue when you use the wrong type of SAS-supplied format to translate the value you have. Second, you need to make sure that you use single quotes around your picture specification. If you do not, SAS will attempt to translate it as a macro call. Lastly, you need to make sure that you use a length specification 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 specification, which is 25 in this case. That is not long enough to accommodate all of the text in the format because each of the format directives are only two characters long, while the values they display are much longer than that. That is why the format length is specified for the third column. The last column shows what you get if you do not specify any length for the XDATE. format. As you can see, all of the values in the fourth column are truncated (at 25 characters.) You can also avoid this problem by explicitly defining a default length in the PICTURE statement with the DEFAULT= option.

```
PROC FORMAT;
PICTURE xdate (DEFAULT=40)
```

The last issue is that even though the dates are displayed as text, the underlying values are numbers, so they are right-justified in the columns. You can use style options in ODS to solve the reporting problem, but there is one more thing to note: you may find yourself with some unwanted leading spaces if you use the formatted date in a concatenated string. However, this problem is easy to avoid by using the CATX(), CATS(), or CATT() functions for string concatenation.

DATES IN TITLES AND FOOTNOTES

Now that we know how to dress up our dates just the way we want them, how can we show them other than in the detail of our reports? For example, if you have a report that is run every week, you could put the date in the title like this:

```
TITLE 'Date of report: March 24, 2014';
```

Unfortunately, that means you will be responsible for changing the code every week. You can get around this by using one of the date and time automatic macro variables in SAS: &SYSDATE; &SYSDATE9; &SYSDAY, &SYSTIME. They are set when the SAS job starts and you cannot change them. If you want to use one of these variables, this is how you would do it:

```
TITLE "Date of report: &SYSDATE9";
/* Since you are using a macro variable, you MUST have DOUBLE quotes around
the text */
```

If you were to run this job on March 24, 2014, this statement would put the text "Date of report: 24MAR2014" at the top of each page. The following day, the title would be "Date of report: 25MAR2014". That is functional, but not very appealing. None of the macro variables is particularly appealing in their native format: &SYSTIME comes out as a 24-hour clock value (e.g., 23:00), while &SYSDATE is the same as &SYSDATE9 with a two-digit year (24MAR09). However, &SYSDAY will look like a proper day of the week (Tuesday).

If that's not exactly what you had in mind, don't worry. You can take advantage of formats and display dates (and times) within your TITLEs and FOOTNOTEs exactly how you want them to look. You can always get the current date and time from SAS using the DATE(), TIME(), and/or DATETIME() functions. It will involve the creation of a macro variable to hold your text, but it takes only a little macro or DATA step coding to do it. Before you put your own date on a page, make sure that you take the default date display off your pages with OPTIONS NODATE;.

Using a DATA Step and CALL SYMPUTX() to Create your Macro Variable

```
DATA _NULL_; /* Don't need to create a dataset */

/* Create a global macro variable called &RDATE and give it the value of
   today's date formatted with the worddate32. format. Use the STRIP()
   function to remove leading spaces or else you'll get an unwelcome
   surprise! Use CALL SYMPUTX instead of CALL SYMPUT to remove any
   trailing blanks in the macro variable. */
CALL SYMPUTX('rdate',STRIP(PUT(DATE(),worddate32.)));
RUN;
TITLE "Date of report: &rdate"; /* Don't forget DOUBLE quotes! */
```

The value of the macro variable &RDATE is "March 24, 2014", and it is left-justified, so the title on each page will now read "Date of report: March 24, 2014". You can take this code as written above, change the format from WORDDATE32. to whatever you need, put it into your reports and your dates will automatically change each day they are run.

The Fancy Example using Custom Formats and Macro Functions

This will show what you can do with custom formats and how you can put them into TITLEs and FOOTNOTEs using SAS macro functions. Once the format is created, this can also be done with a DATA step as shown above. The first part of the example below creates a custom format named DEDATE using the PICTURE statement.

```
PROC FORMAT;
PICTURE dedate
. - .z = "No Date Available"  /* What if the date (datetime in this case)
                                 is missing? */
LOW - HIGH = '%B %d, %Y at %I:%0M %p' (DATATYPE=DATETIME)
;
RUN;

/* Now use the %SYSFUNC() function to access a DATA step function in macro
language */
%LET rdate=%SYSFUNC(DATETIME(),dedate32.);
TITLE "Date of report: &rdate"; /* Don't forget DOUBLE quotes! */
```

The FORMAT procedure uses a mixture of text and date directives to create the display. Line 3 is there in case a datetime value is missing (if you use the DATETIME() function, it will never be missing.) Line 4 contains the date directives as well as text that will be printed with the date directives, but the most important part of the line is the DATATYPE= argument. This is not optional, because it tells the format what type of value to expect so that it can be translated correctly. The value of the DATATYPE argument can be DATE, TIME, or DATETIME. Sending the wrong type of data to a custom format will give you incorrect results just like sending the wrong type of data to a SAS-supplied format does.

Line 9 demonstrates a nice feature of the %SYSFUNC function: you can format the result of a call without needing to use the PUT() or PUTN() function, so you can just tell the macro processor the format you want to use without having to nest %SYSFUNC or %QSYSFUNC calls. You will need to specify the length of the format because its default length is only 22 (the number of characters between the quotes in line 4.) This also automatically justifies the formatted result properly within the macro variable &RDATE without having to use the SAS autocall macro %LEFT() to left-justify the result and store it in the macro variable &RDATE. Our report title will now say, *Date of report: March 24, 2014 at 4:08 PM* (italics mine, not actual appearance) as per the date directive in line 4. The only caution is that your title will be updated each time you execute the code that creates the macro variable. If you do not want the title line to update throughout your report, make sure you only execute the code once, and do it at the beginning of your program.

READING DATES AND TIMES AND DATETIMES

So far, our examples have all used date constants, but you cannot 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. You will need both the formatted INPUT statement and an informat in order to read date, time, or datetime data from a flat file. Here is 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 the above file, you would use the following DATA step. Note the MMDDYY10. after the variable name SAMPLE_DATE. This is the informat, and it tells SAS how to process the ten characters it is reading (that's what the 10 in MMDDYY10. means.)

```
DATA read_dates;
INFILE "a_few_dates.txt" PAD;
INPUT @1 sample_date :MMDDYY10.;
RUN;
```

Here is the output. The first column is the value that is stored in the dataset created by the above code. Extra columns have been added to show that value when it is displayed using two different formats.

SAS Date Value	Formatted Using MMDDYY10.	Formatted Using WEEKDATE.
20022	10/26/2014	Sunday, October 26, 2014
19239	09/03/2012	Monday, September 3, 2012
2690	05/14/1967	Sunday, May 14, 1967
20325	08/25/2015	Tuesday, August 25, 2015
19905	07/01/2014	Tuesday, July 1, 2014
18702	03/16/2011	Wednesday, March 16, 2011
4092	03/16/1971	Tuesday, March 16, 1971
3015	04/03/1968	Wednesday, April 3, 1968
38254	09/25/2064	Thursday, September 25, 2064

Result 5. Using an Informat to process a file.

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 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. It's easy to get the wrong result 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 would not give you any message that anything was abnormal until the middle two digits in the data field were greater than 12. Let's see what happens when we use the wrong informat with the same file:

```
DATA read_dates;
INFILE "c:\book\examples\a_few_dates.txt";
INPUT @1 sample_date :DDMMYY10.;
RUN;

NOTE: Invalid data for sample_date in line 1 1-10.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6-
---+----7----+----8----+--
1         10/26/2014 10
```

```
sample_date=. _ERROR_=1 _N_=1
NOTE: Invalid data for sample_date in line 3 1-10.

3         05/14/2012 10
sample_date=. _ERROR_=1 _N_=3
NOTE: Invalid data for sample_date in line 4 1-10.
4         08/25/2015 10
sample_date=. _ERROR_=1 _N_=4
NOTE: Invalid data for sample_date in line 6 1-10.
6         03/16/2011 10
sample_date=. _ERROR_=1 _N_=6
NOTE: Invalid data for sample_date in line 7 1-10.
7         03/16/1971 10
sample_date=. _ERROR_=1 _N_=7
NOTE: Invalid data for sample_date in line 9 1-10.
9         09/25/2064 10
sample_date=. _ERROR_=1 _N_=9
```

At least you can see the error messages. But you did not get an error message on every line. What happened to the data on lines 2, 5 and 8? Let's show the result side by side with the table above on the right in italics.

SAS Date Value	Formatted Using MMDDYY10.	Formatted Using WEEKDATE.		*SAS Date Value*	*Formatted Using MMDDYY10.*
				20022	*10/26/2014*
19061	03/09/2012	Friday, March 9, 2012		*19239*	*09/03/2012*
				2690	*05/14/1967*
				20325	*08/25/2015*
19730	01/07/2014	Tuesday, January 7, 2014		*19905*	*07/01/2014*
				18702	*03/16/2011*
				4092	*03/16/1971*
2985	03/04/1968	Monday, March 4, 1968		*3015*	*04/03/1968*
				38254	*09/25/2064*

Result 6. Using an incorrect format for your data compared to using the correct one.

You can see how important it is to use the correct informat; instead of the data you expect, you get missing values and incorrect data.

How Do I Use an Informat When My Raw Data Is Delimited?

The first example used a column pointer (the "@1") to tell SAS where to start reading, because the data are neatly aligned in columns throughout the flat file.

```
INPUT @1 sample_date :MMDDYY10.;
```

What do you do when you have a file that doesn't use columns, but has delimiters between fields such as tab characters or commas? Start the informat name with a colon (":"). Let's look at some data from a CSV file:

```
Smith,12DEC1950,Goal,1989
Jones,30OCT1994,Defense,
Hull,9AUG1964,Wing,2005
Oates,27AUG1962,Center,2004
```

As you can see, the obvious date isn't in the same column for each line. The following code is used to read these data. The DELIMDATES dataset is then used with PROC REPORT to produce the resulting table.

```
DATA delimdates;
INFILE "delim.csv" DLM=',' DSD PAD MISSOVER;
/* MISSOVER translates missing columns into missing data */
INPUT name $ birthDate :date. position $ retired;
RUN;
```

The bolded ":date." in line 3 tells SAS to apply the DATE. informat to the second field, since we can't figure out where the field starts. Here's the result, again using an aliased variable to provide a formatted version of the variable birthDate.

Name	Birth Date (SAS Date Value)	Formatted Birth Date	Position	Year Retired
Smith	-3307	Tuesday, December 12, 1950	Goal	1989
Jones	12721	Sunday, October 30, 1994	Defense	
Hull	1682	Sunday, August 9, 1964	Wing	2005
Oates	969	Monday, August 27, 1962	Center	2004

Result 7: Using the colon (:) informat modifier to read delimited data

READING DATES AND TIMES AND DATETIMES THE EASY WAY

Version 9 of SAS addressed one major 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.

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, but the amount of extra processing depends on how many dates you need to process. What may be negligible for small amounts of data could become significant in a multi-billion record data warehouse with multiple dates to handle.

Overhead aside, doesn't this solve all of your problems with dates and informats? Unfortunately, no. What if you have to read a value such as "06/11/2015"? Or even worse, "06/11/15"? Is it June 11, 2015, November 6, 2015, or November 15, 2006? There is an option that will allow you to tell SAS what it should be. The DATESTYLE option comes in four flavors:

MDY	Sets the default order as month, day, year. "06-11-15" would be translated as June 11, 2015
DMY	Sets the default order as day, month, year. "06-11-15" would be translated as November 6, 2015
YMD	Sets the default order as year, month, day. "06-11-15" would be translated as November 15, 2006
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, "06-11-15" would be translated as June 11, 2015.

Table 2: Values for the DATESTYLE option

Although it may now seem simple to resolve ambiguous dates, it is not foolproof. If you use the ANYDT series of informats for date processing, always check the resulting data, especially if you need to process dates with two-digit years. However, in general, it is good programming practice to check your datasets regardless of how simple the task of creating them may seem.

AM I READING ENOUGH?

One last 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. However, this default may not match your data.

The bottom line is that when you are reading formatted date values that have not been stored as SAS dates, it is best to look at the data that you have produced before you actually use it.

CREATING DATES, TIMES AND DATETIMES FROM NUMBERS

There are three functions in SAS that create SAS date, time, and datetime values from 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. All the arguments to these functions must be numeric; they can be actual values or numeric variables. In the DHMS() function, the date used can be a date literal, a SAS date value, or a numeric variable.

Month, day and year are the arguments to the MDY() function as follows: MDY(***month,day,year***)***;*** This function calculates a SAS date value as long as: 1) none of the arguments are missing; 2) month is between 1 and 12, and 3) the combination of month, day, and year is a valid calendar date. As an example of the last, MDY(2,31,2014); will return a missing value because February 31, 2014, does not exist.

The HMS(***hours,minutes,seconds***)***;*** function creates a SAS time value from the hours, minutes, and seconds given, and once again, all three values must be non-missing. When you use this function, it is important to remember that there are two distinct concepts of time: 1) clock time, and 2) elapsed time. Because of that, the only restriction on this function is that ***hours*** must be greater than or equal to zero, while ***minutes*** and ***seconds*** have no restrictions on their value. HMS(7,45,80); will calculate to 27,980 seconds; to SAS, that is exactly the same as HMS(7,46,20). The format that you use ultimately determines the context of the time value. If you are talking about clock time, the value returned will be MOD(result,86400). You can use the following program to test this and help your understanding, substituting actual non-clock values for ***hours, minutes,*** and ***second***. The first value printed in the resulting log will be the total number of seconds; the second value will display the number of seconds calculated to a 24-hour clock (that MOD(value,86400) thing again.)

```
DATA _NULL_;
x = HMS(hours,minutes,seconds);
PUT x= +3 'x=' x time.;
RUN;
```

The DHMS(***date,hour,minute,second);*** function will create a datetime value from the component parts of ***date, hour, minute,*** and ***second*** in the same way the previous two functions work. Again, each argument must not be missing. ***date*** can be a date literal, a SAS date value, or a numeric variable.

EXTRACTING MONTH, DAY, YEAR, ETC., FROM YOUR SAS DATE

There are also functions that will pull the individual component values from SAS date, time and datetime values. The list is below, and ***arg*** represents the appropriate date, time or datetime value for the function.

DATEPART(arg)	Extracts the date from a SAS datetime value as a SAS date value.
DAY(arg)	Extracts the number of the day of the month from a SAS date value.
HOUR(arg)	Extracts the hour from a SAS time value.
MINUTE(arg)	Extracts the minute from a SAS time value.
MONTH(arg)	Extracts the month from a SAS date value.
SECOND(arg)	Extracts the second from a SAS time value.
TIMEPART(arg)	Extracts the time portion from a SAS datetime value as a SAS time value.
WEEKDAY(arg)	Extracts the number of the day of the week, where Sunday=1, Monday=2, etc. from a SAS date value.
YEAR(arg)	Extracts the year from a SAS date value. Since obtaining the year value is the point of using this function, it is important to remember that the YEARCUTOFF option affects two-digit years.

Table 3: Functions to get components from a SAS date, time, or datetime value

You need to make sure that you use the appropriate function for the type of value you are working with. For example, do not try to use the DAY() function to extract a day from your datetime value. It will not work correctly, 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 after March of 1960, SAS will give you an error message. However, here the error message does not occur because SAS recognizes it as a datetime value; rather it occurs because it is trying to determine a day for a value that is too large for the function to work. In that vein, here is 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? Isn't that the correct value for the datetime value? 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 86,400 (the number of seconds in a day) just cycles into the next day. Therefore, when you calculate the minutes from the datetime value as shown above you may seem to get the right answer, when in reality, you are getting the result of MOD(value,86400). So while the time extraction functions seem to work with datetimes, it is not good practice. You can also clearly see that trying to get the minute out of the date value gives you an incorrect result.

The YEAR() function is affected by the YEARCUTOFF option only if you have a date literal with a two-digit year, and this can lead to unexpected results. On the other hand, if you are extracting the year from a SAS date value, the YEARCUTOFF option will have no effect, because the concept of two-digit year vs. four-digit year does not exist; it is simply the number of days since January 1, 1960.

INTERVALS

Intervals are another way people track periods of time. SAS can allow you to manipulate dates and times and datetimes by these intervals. SAS Intervals 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 defined in it. The intervals represent periods of time such as days, weeks, months and quarters, to name a few. A complete list of the SAS-supplied intervals, along with the default starting point for each one, is in Appendix 3. While intervals such as "YEAR" and "MONTH" may seem self-explanatory, if you just jump right in and use SAS intervals without reading about them first, you may not get the results you expect.

WHEN IS A YEAR NOT A YEAR?

The syntax for the INTCK() function is: **INTCK(*interval,start-of-period,end-of-period,method*); *interval*** must be in quotes. This function calculates the number of ***intervals*** between the dates represented by ***start-of-period*** and ***end-of-period***. ***method*** is either CONTINUOUS or DISCRETE, and must also be enclosed in quotes. If you do not specify ***method***, DISCRETE is the default. The CONTINUOUS method can best be summed up as counting intervals the way we intuitively think about periods of time such as months or years. For the purposes of this paper, however, we will focus on the DISCRETE method, which is the way that the INTCK() function has calculated intervals since its creation. There is a great deal of existing code that relies on the DISCRETE method, and I strongly advise against changing it to the more intuitive method.

To illustrate the potential for problems with SAS intervals, consider the following program:

```
DATA _NULL_;
v1 = INTCK('YEAR','01jan2016'd,'01jan2017'd);
v2 = INTCK('YEAR','31dec2016'd,'01jan2017'd);
PUT v1= +3 v2= +3;
RUN;
v1=1    v2=1
```

Now wait a minute. We know that a year from December 31, 2016, is not January 1, 2017. What happened? SAS intervals are not a shortcut to doing the math. When the DISCRETE method is used (either explicitly or by default,) 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*. The YEAR interval will start on January 1, 2016 for any date in 2016. Therefore, given any starting date in 2016, INTCK() will only count a single YEAR interval for any given date in 2017 because it only passes January 1, 2016 one time. There is an enormous potential for bad results if you misunderstand how INTCK() calculates using its default (the DISCRETE) method. The important thing to remember about intervals is that they are based on the starting point of the interval for the ***start-of-period*** date given to the function, and not the ***start-of-period*** date itself.

The INTNX() function advances a given date, time or datetime by a specified number of intervals. The syntax for this function is **INTNX(*interval,start-date,number-of-increments,alignment*);**, where ***interval*** is one of the SAS intervals from the previous page (again in quotes)**,** ***start-date*** is the starting date, and

number-of-increments is how many intervals you want to change the date. ***alignment*** will adjust the result of INTNX() relative to the interval given. It can be 'B', 'M', or 'E' (quotes necessary) for the beginning, middle, or end of the interval, respectively. There is also the 'S' alignment value, which will adjust the result to the same day as given in the ***start-date*** argument. To illustrate how INTNX() and alignment works, here is a sample program that adds six months to March 20, 2017, with the result in bold.

```
DATA _NULL_;
a = INTNX('MONTH','20MAR2017'd,6);
b = INTNX('MONTH','20MAR2017'd,6,'B');
c = INTNX('MONTH','20MAR2017'd,6,'M');
d = INTNX('MONTH','20MAR2017'd,6,'E');
e = INTNX('MONTH','20MAR2017'd,6,'S');
PUT "6 months from 3/20/2017 with default alignment = " a mmddyy10.;
PUT "6 months from 3/20/2017 aligned with beginning of MONTH interval= " b
mmddyy10.;
PUT "6 months from 3/20/2017 aligned with middle of MONTH interval= " c
mmddyy10.;
PUT "6 months from 3/20/2017 aligned with end of MONTH interval= " d
mmddyy10.;
PUT "6 months from 3/20/2017 aligned with same day in MONTH interval= " e
mmddyy10.;
RUN;
```

```
6 months from 3/20/2017 with default alignment = 09/01/2017
6 months from 3/20/2017 aligned with beginning of MONTH interval= 09/01/2017
6 months from 3/20/2017 aligned with middle of MONTH interval= 09/15/2017
6 months from 3/20/2017 aligned with end of MONTH interval= 09/30/2017
6 months from 3/20/2017 aligned with same day in MONTH interval= 09/20/2017
```

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 it, SAS seems to have 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**".

It is important to remember that 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 are moving the start of the year to the second month. If you do not account for this, you will 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.

ISO 8601 AND SAS DATES, TIMES, AND DATETIME VALUES

For us long-time SAS users (pre-version 9), making dates, times and datetimes adhere to the ISO 8601 standard required a little bit of ingenuity. There are macros that have been constructed to take the ISO representations and turn them into SAS-appropriate values, and to transform SAS values for date, time, and datetimes into the ISO 8601 representation. If you are using one of those macros, and you now have access to SAS version 9 or higher (the ISO 8601 formats and informats were first introduced in SAS 8.2, but version 9 has several enhancements), you can stop now. For the sake of brevity, this paper will only discuss the conversion of dates, times, and datetimes between SAS and their ISO 8601 extended representations, and not the more complicated concept of describing periods.

To convert an ISO 8601 datetime value (which is a character variable) into its SAS equivalent and its components, you just need three lines of code:

```
SAS-datetime-variable = INPUT(ISO 8601-datetime-variable),E8601DT.);
SAS-date-variable = INPUT(ISO 8601-datetime-variable),E8601DA.);
SAS-time-variable = TIMEPART(SAS-datetime-variable);
```

That's it, unless the ISO 8601 datetime value includes time zones, in which case the datetime conversion becomes:

```
SAS-datetime-variable = INPUT(ISO 8601-datetime-variable,E8601DZ.);
```

If you are using a version of SAS prior to 9.4, here's a necessary alternative:

```
SAS-datetime-variable = INPUT(COMPRESS(ISO 8601-datetime-variable,'-:.+ '),
B8601DT.);
```

Why do you need to use the COMPRESS() function? It's a solution to a problem described in SAS Note 43337, which was fixed in SAS 9.3 TS1M1. Prior to that release, if you had a partial ISO date value (e.g., "2011-03-20T15:30"), trying to process it with the B8601DT. informat would not work because the default length of the informat is 19 (which is also the minimum length allowed by SAS), but the length of the character value is 16. You cannot specify B8601DT16. (if you tried, you received an error.) Conversely, if you use the default length, you will get the error message: "**NOTE: Invalid argument to function INPUT**" in your SAS log.

There is one downside to using the basic (starts with 'B", not "E") ISO datetime informats: SAS will make assumptions for partial ISO date, time, and datetime values. This can be tricky when it comes to processing partial ISO datetime values where the date and/or time may not be complete. SAS will not return a missing value, even if that is what you want. For missing times, SAS will substitute midnight; for missing days, the first, and for missing months, January. If there is no year present, then the datetime value will be set to missing. The workaround is easy. You can conditionally execute the datetime code above by checking the ISO string for the letter 'T'. If it is present, then the code will provide you with a datetime value. If not, then set it to missing, and calculate the date as follows:

```
SAS-date-variable = DATEPART(INPUT(COMPRESS(ISO 8601-date-variable,'-:. ')
,B8601DT.));
```

It is even easier to represent a SAS datetime value using the ISO 8601 standard: you simply use one of the following formats:

E8601DA.	Represents a SAS date value in the format *yyyy-mm-dd*
E8601DT.	Represents a SAS datetime value in the format *yyyy-mm-ddThh:mm:ss.ffffff* (*ffffff* represents fractional seconds)
E8601DZ.	Represents a SAS datetime value in the format *yyyy-mm-dd*T*hh:mm:ss*+|-*hhmm*
E8601TM.	Represents a SAS time value in the format *hh:mm:ss.ffffff* (*ffffff* represents fractional seconds)

EXCEL

We cannot overlook the potential for incorrect results whenever you convert from Microsoft Excel into SAS or vice versa. Like SAS, Excel counts days from a fixed reference point, and it uses January 1, 1900. Excel keeps track of times as decimal fractions where you can multiply by 24 (hours) to get the clock time of the day. Six a.m. is a quarter of a day (.25); noon is exactly one-half day; and so on. A datetime value in Excel is the sum of the day and the time value. For example, September 5, 2014 is 41887 to Excel, so noon on September 5, 2014 would be 41887.5.

Exchanging historical dates between SAS and Excel is a definite problem. While Excel uses a zero point of January 1, 1900, unlike SAS, it cannot count backwards from zero. Therefore, any date prior to 1900 in Excel is represented as text, not as an Excel date value. What do you do if you have dates before January 1, 1900, to convert? Going to Excel, you will have to store them as character strings, and you will not be able to use any of the Excel math or date 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. If you use the IMPORT wizard and/or procedure, you may get the dreaded dot, or worse. SAS may decide the entire column is a character variable, leaving you with text where your date values should be. Unless you have historical dates, the IMPORT procedure or SAS/ACCESS® will convert dates, times and datetimes for you, and it is fairly reliable. However, when you are importing into SAS from any other software or database, always test your dates, times and datetimes, because different software uses different representations of dates, times, and datetimes, and the default conversion is not foolproof.

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 data. However, that flexibility can also make it easy to make mistakes. Dates, times, and datetimes are stored as numbers in SAS. 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 one, and each second that passes increments the time and datetime counters by one.

SAS dates and times are displayed by using formats. There are many SAS-supplied formats and each 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 SAS-supplied formats meets your needs, you can create your own, either by defining the display to correlate with certain values, or by defining the display with a template using symbolic directives. You can use formats to customize titles and footnotes in your output with date and time information.

We use informats to translate dates and times as we know them into the values recognized by SAS. As with formats, there are many SAS-supplied 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 do not know the format of the alphanumeric characters you will be processing, you can use the ANYDATE informats, but be prepared for your job to take longer, and always check your results.

You may extract the individual components of a SAS date, time, or datetime value, such as month number, day or hour. However, you need to make sure 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 non-missing for these functions to work.

It is frequently useful to refer to periods of time by familiar references such as year or quarter. SAS uses intervals to handle this. 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 by default, these functions use SAS intervals by measuring the intervals from the start point of the interval, not the dates you supply.

There are many more uses of SAS dates, more functions relating to dates and times and datetimes, and more capabilities of intervals than have been shown in this paper. This introduction provides an overview of handling and manipulating dates and times in SAS, and addresses a large percentage of date- and time-related tasks. Should you need more than what is in this paper, the tools SAS provides to handle dates and times should allow you to accomplish your tasks.

REFERENCES

Morgan, Derek P. 2014. *The Essential Guide to SAS® Dates and Times, Second Edition*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION:

Comments and questions welcome to:

Derek Morgan
E-mail: mrdatesandtimes@gmail.com

APPENDIX 1: HOW DO I?

1) Convert a character date into a valid SAS date?

This is one of the most frequently asked questions, usually occurring when moving data from other software into SAS. The example uses the ANYDTDTE. informat, but it is more efficient to use one of the specific date informats.

```
SAS-date-variable = INPUT(character_date,ANYDTDTE.);
```

2) Put a date into a file name?

Ever been asked to label a file with a date? This example puts the full month name and four-digit year in the name of the Excel file.

```
PROC FORMAT LIB=LIBRARY;
PICTURE monthyr (DEFAULT=15)
LOW-HIGH = '%B %Y' (DATATYPE=DATE)
;;;;
RUN;
%LET datestr=%SYSFUNC(STRIP(%SYSFUNC(DATE(),monthyr.)));
ODS TAGSETS ExcelXP FILE="&datestr Monthly Report.xml";
```

With a little more effort on your own, you can label worksheet tabs in a workbook using this method.

3) Turn my date from a number into something I understand?

Use a format. In a DATA step or procedure, the code would be:

```
FORMAT date mmddyy10.;
```

In SQL:

```
SELECT date FORMAT=mmddyy10.
```

4) Turn my asterisks into a date? I used a date format like you said.

This generally means the value you are trying to display is too big to display with the format you are using. With dates, this happens most often when you use a date format to display a datetime value. The log segment below demonstrates:

```
1   DATA _NULL_;
2   datetime=1682340217;
3   PUT "Datetime value formatted with DATE9. format " datetime date9.;
4   PUT "Datetime value formatted with DATETIME19. format " datetime
datetime19.;
5   RUN;
Datetime value formatted with DATE9. format *********
Datetime value formatted with DATETIME19. format 23APR2013:12:43:37
```

5) Figure out how many days have elapsed between two SAS dates?

Since SAS keeps track of dates as number of days relative to January 1, 1960, subtraction is the quickest method. This also works for times and datetimes, but you will get the result in seconds.

```
how_many_days = date2 - date1;
```

6) Create a datetime from month, day and year when I don't have a time?

This takes two steps if you don't already have a SAS date:

```
SAS_date = MDY(month,day,year);
SAS_datetime_value = DHMS(SAS_date,0,0,0);
```

If you have a SAS date and SAS time available you can just use both in the DHMS() function. SAS times are stored in seconds, so there's no need to convert into hours, minute, seconds, but you do need zeroes as placeholders for hours and minutes:

```
SAS_datetime_value = DHMS(SAS_date,0,0,SAS_time);
```

APPENDIX 2: HANDY CUSTOM DATE FORMATS AND INFORMATS

Here are some of the most common date formats and informats I've needed. I create a permanent format library for my projects, so I don't have to worry about coding these every time I need one of them.

```
PROC FORMAT LIB=LIBRARY;

INVALUE mthct (UPCASE) /* translate month abbreviations into month numbers
*/
'JAN' = 1
'FEB' = 2
'MAR' = 3
'APR' = 4
'MAY' = 5
'JUN' = 6
'JUL' = 7
'AUG' = 8
'SEP' = 9
'OCT' = 10
'NOV' = 11
'DEC' = 12
' ', . = .
OTHER = _ERROR_
;;;;

VALUE ctday /* Translate weekday numbers (like from the WEEKDAY() function)
               Into weekday names
*/
1 = "Sunday"
2 = "Monday"
3 = "Tuesday"
4 = "Wednesday"
5 = "Thursday"
6 = "Friday"
7 = "Saturday"
;;;;

INVALUE ctday /* Translate weekday names into weekday numbers */
"Sunday" = 1
"Monday" = 2
"Tuesday" = 3
"Wednesday" = 4
"Thursday" = 5
"Friday" = 6
"Saturday" = 7
OTHER = _ERROR_;
;;;;

PICTURE monthyr (DEFAULT=15) /* Display date as full month name and 4-digit
year */
LOW-HIGH = '%B %Y' (DATATYPE=DATE)
;;;;
RUN;
```

APPENDIX 3: SAS INTERVALS AND THEIR STARTING POINTS

Interval Name	Definition	Default Starting Point
DAY	Daily intervals	Each day
WEEK	Weekly intervals of seven days	Each Sunday
WEEKDAY*days*W	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	1st and 16th of each month
MONTH	Monthly intervals	1st 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
DTWEEKDAY*days*W	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	1st and 16th of each month
DTMONTH	Monthly intervals used with datetime values	1st 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 time values	Seconds
MINUTE	Minute intervals used with time values	Minutes
HOUR	Hourly intervals used with time values	Hours