It's Five O'Clock Somewhere!!!
Handling Dates And Times In SAS®

Toby Dunn, AMEDDC&S (CASS), San Antonio, TX
Sarah Woodruff, Westat Inc., Rockville, MD

ABSTRACT
Dates and Times - we use these concepts every day and they are strewn throughout our SAS® code. So what about dates and times cause us problems? Why do they cause so many programmers, both new and seasoned, headaches? More importantly, do they have to be so hard to deal with? We are here to say "No, they don't." In this paper we will take you through the ins and outs of working with SAS date, datetime, and time values. Once armed with a range of concepts and functions you will come to see just how easy it is to work with date and time values in SAS, whether used in open or macro code. Once you have mastered these skills you too will be singing "It's Five O'Clock Somewhere" and looking to kick back with your favorite beverage.

Keywords: Date, Time, DateTime,

Introduction
As long as man has been around on earth he has tried to measure from one moment to the next. This helps give some sense of order to what was, is, and will be coming in his future. It helped man track the seasons and even know when food supplies would be available. More recently with the advancement of technology it allows him to measure such things as the air brake controls on the Bugatti Veyron in milliseconds to ensure that the car slows down before the break pads are completely worn out.

Humans started counting time with simple things like the number of full moons that occurred between events. As man evolved, there was the need for greater precision in their measurements and a system of record-keeping. One of the earliest tracking tools was the use of basic calendars. Throughout history humans have used many different calendars and while some countries (like China) use their own calendar, the most widely used today is the Gregorian.

Historical Reference and SAS Dates/Times
The Gregorian calendar is part historical and part man made. It consists of 365.25 days on average; this occurs because a typical year consists of 365 days and a "leap years" has 366 days, with the additional day occurring every four years. It is further complicated by the fact that any year divisible by 100 but not by 400 is not considered a leap year.

In order for computers and computer programmers to handle these date and time values, computers need a way to easily determine what date or time the value represents, be able to adjust that date/time value, and finally be able to print it out in way that humans can easily understand. To accomplish this computers use numeric scalars. Computers love numbers and as long as there is a reference point, such as a denoted numeric scalar for a base date or a real number for time values which can then represent a SAS® Date/Time value of 0. With this established, adding or subtracting values becomes easy and all that is needed is some way to format the output in a human readable way.

The base SAS Date/Time value is January 1, 1960. This date was chosen by SAS co-founder, Tony Barr, because he wanted a date value that would not only predate most of the card data (remember SAS originally only ran on the mainframe), but was also recent enough to allow timestamps to fit into less storage space.

SAS Date values are incremented by one for each day after January 1, 1960 and minus one for each day before it. The valid range of values for SAS Dates are January 1, 1582 up through December 21, 20000. Any value containing a SAS DateTime/Time value increases from this point by 1 for each second from midnight of that baseline date. So Date values contain integers representing the number of days while Datetime/Time values contain real numbers representing the number of seconds from the reference date.

Date/Time Literals
One of the easiest methods to get a Date/Time value in SAS is simply to declare a human readable value as a Date/Time value. SAS has made this relatively painless by allowing Date/Time values enclosed in single or double

quotes and postfixed with a D (dates values), T (Time values) or DT (Datetime values). The only thing that one has to ensure is the use of the appropriate post fix operator for the value you are specifying. For example, you can't use DT for a Date only or Time only value.

```
Data _Null_ ;
       Date     = '01Jan2010'D ;
       Time     = '01:30:15'T  ;
       DateTime = '01Jan2010:01:30:15'DT ;
Put Date=
    Time=
    Datetime= ;
Put Date= Date9.
    Time= Time.
    DateTime= DateTime18. ;

Run ;

Date=18263    Time=5415     DateTime=1577928615
Date=01JAN2010 Time=1:30:15       DateTime=01JAN10:01:30:15
```

Here we created all three of the Date/Time literal values. The Date literal has the day, a three character month, and a four character year (it could also be two characters). The Time value has the hours, minutes, and seconds each separated by a colon. Finally, the DateTime value has the date separated by a colon from the time portion.

The first PUT statement shows the internal numeric scalar and real number in which SAS holds these values. While they can look confusing at first, they are in fact the easiest way to hold and deal with these values in any computer programming language. The second PUT statement uses formats to translate these numeric scalars back into human readable form. It is important to note that while it is best to hold Date/Time values as numeric scalars, unless you specify a format, SAS will not automatically translate them back to a readable form for you.

Before we move on it is an appropriate time to discuss the optimal size of the variable to hold these values. As we know Date/Time values are numeric variables and as such SAS defaults these to a length of 8. However, since SAS Date values are numeric scalars and the upper bound of the Date range in SAS is so far out it is doubtful any of us will use it, one could hold the a SAS Date value with a length of 4. This would allow the valid date range to span from January 1, 1582 through October 23, 7701.

Time values in SAS are held as real numbers due to the fact that it may need to hold fractions of a second. This necessitates that the value be held in no less than a length of 4 (if you do not care about the fractions of a second) and at least 8 if you need that level of precision. DateTime values require a minimal length of 6 which will hold the date and time portion minus the fractions of a second. If the fractions of a second are important then use a length of 8. What is crucial to note is that when you start shrinking the size of the variable holding your Date/Time values below the recommended levels you will lose precision.

Non-Arithmetic Date/Time Functions

Once one gets past simply putting Date/Time values into SAS it then becomes necessary to be able to manipulate them. Here again SAS has made it relatively painless by providing a whole host of functions with which to work. Date/Time functions come in two varieties: those which do arithmetic and those that do not. Of those that do not we can easily group these into three categories: those that create a Date/Time value without using another SAS variable, those that take other variables' values and create a Date/Time value, and those that extract some part of a Date/Time value.

Some of the most commonly used Date/Time functions are those that create a Date/Time value in SAS but do not require any other input. In this section we will cover the following functions: Date, Today, DateTime, and TimePart.

Date Function

The Date function returns the current date from the OS on which SAS is running.

```
Data _Null_ ;
       Date = Date() ;
       Put Date= Date9. ;
Run ;
```

```
Date=11MAY2009
```

Today Function
The Today function does the same thing as the Date function, returning the current date as a SAS Date value.

```
Data _Null_ ;
       Today = Today();
       Put ToDay= Date9. ;
Run ;
```

```
Today=11MAY2010
```

Time Function
The Time function returns the current time.

```
Data _Null_ ;
       Time = Time() ;
       Put Time= Time. ;
Run ;
```

```
Time=11:35:25
```

DateTime Function
The DateTime functions returns the current Datetime value.

```
Data _Null_ ;
DateTime = DateTime() ;
Put DateTime= Datetime18. ;
Run ;
```

```
DateTime=11MAY10:11:35:25
```

The second group of Date/Time functions we would like to present is those that take another SAS Date/Time value as its input and returns a part of that back to the calling environment. These functions are very handy when you need to compare only part of Date/Time value to some known value or when you need to extract part of these values to use later. In this section we will cover the following functions: Day, Week, WeekDay, Year, QTR, DatePart, Time, and TimePart.

Day Function
The Day function requires a valid SAS Date variable as its argument and returns the numeric day value that is represented by the argument. This function will produce a number between 1 and 31.

```
Data _Null_ ;
       Date = '01Jan2010'd ;
              Day  = Day( Date ) ;

       Put Day= ;
Run ;
```

```
Day=1
```

Week Function
The Week function returns the number of the week in which the day of the Date value given as the first argument occurs.

Week(SAS Date , Modifier)
Modifiers:
U specifies the SAS date value by using the number-of-week within the year (Sunday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0-53, using a leading zero with a maximum value of 53. This is the default modifier if none is given explicitly.

V specifies the SAS date value. The number-of-week value is represented as a decimal number in the range 01-53, using a leading zero with a maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year.

W calculates the SAS date value by using the number of week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0-53, using a leading zero with a maximum value of 53.

Here are examples with and without modifiers:

```
Data _Null_ ;
       Week1 = Week( '01FEB2003'd ) ;
       Week2 = Week( '01FEB2003'd , 'U' ) ;
       Week3 = Week( '01FEB2003'd , 'V' ) ;
       Week4 = Week( '01FEB2003'd , 'W' ) ;
              Put Week1=
                  Week2=
                  Week3=
                  Week4= ;
Run ;

Week1=4 Week2=4 Week3=5 Week4=4
```

WeekDay Function
The WeekDay function returns the day of the week from the SAS Date value in its argument. The valid range of values are 1-7 where 1 represents Sunday, 2 represents Monday through 7 representing Saturday.

```
Data _Null_ ;
       WeekDay = WeekDay( '01Jan2010'd ) ;
       Put WeekDay= ;
Run ;

WeekDay=6
```

Year Function
The Year function returns the year that is associated with the Date value supplied in its argument.

```
Data _Null_ ;
       Year = Year( '01Jan2010'd ) ;
       Put Year= ;
Run ;
```

```
Year=2010
```

QTR Function
The QTR function returns the quarter of the year that is associated with the Date value supplied in its argument.

```
Data _Null_ ;
       QTR = QTR( '01Jan2010'd ) ;
       Put QTR= ;
Run ;
```

```
QTR=1
```

DatePart Function
The DatePart Function returns the Date portion from an argument that is a DateTime value.

```
Data _Null_ ;
       DatePart = DatePart( '01Jan2010:01:12:30'dt ) ;
       Put DatePart= Date9. ;
Run ;
```

```
DatePart=01JAN2010
```

Note that the value returned needs to be formatted. This is because the function returns the date portion of the DateTime value as a SAS Date value.

TimePart Function
The Timepart function returns the Time portion of a DateTime value as a SAS Time value.

```
Data _Null_ ;
       TimePart = TimePart( '01Jan2010:01:12:30'dt ) ;
       Put TimePart= Time. ;
Run ;
```

```
TimePart=1:12:30
```

This works beautifully if you need to use the current date/time values in your system clock or if you already have a date/time value and need a portion of it. However, what about a situation where you have pieces of a date/time value in multiple variables and need to create a SAS date/time value? SAS has this covered as well with the following functions: MDY. DHMS, and HMS functions.

MDY
The MDY (month, day, year) function takes the month (a number from 1-12), day (a number from 1-31), and year (in two or four digits) values stored in other variables and creates a SAS date value with them.

```
Data _Null_ ;
Day   =  1    ;
Month =  3    ;
Year  =  1999 ;
Date= MDY( Month , Day , Year ) ;
Put Date= Date9. ;
Run ;
```

```
Date=01MAR1999
```

DHMS
The DHMS function returns a SAS DateTime value by combining a SAS Date value with hours , minutes, and seconds. These must be numeric values.

```
Data _Null_ ;
      Date   = '01Jan2010'd ;
      Hour   = 3  ;
      Minute = 20 ;
      Second = 12 ;
      DateTime= DHMS( Date , Hour , Minute , Second ) ;
      Put DateTime= DateTime18. ;
Run ;
```

```
DateTime=01JAN10:03:20:12
```

HMS
The HMS function returns a SAS Time value from supplied hour, minute, and second numeric values.

```
Data _Null_ ;
      Hour   = 3  ;
      Minute = 20 ;
      Second = 12 ;
      Time= HMS( Hour , Minute , Second ) ;
      Put Time= Time. ;
Run ;
```

```
Time=3:20:12
```

Arithmetic Date/Time Functions
In the previous section we looked at Date/Time functions that created Date/Time value or extracted parts from a DateTime value. What if you needed to determine how old someone is or how many days or years have transpired since someone was hired to determine if they are eligible for a raise? To do this you would need some way to manipulate these Dates/Time values. SAS has delivered once again by producing the DatDif, YrDif, Intck, and Intnx functions.

We know from the opening sections of this paper that SAS Dates are the number of days from January 1, 1960 and DateTime values are the number of seconds from January 1, 1960. So armed with this and a little math we can do our necessary calculations.

First, we will increase a date and DateTime value by one year.

```
Data _Null_ ;
      StartDate     = '01Jan2010'd ;
      StartDateTime = '01Jan2010:01:12:30'dt ;
      NewDate     = StartDate + 365.25 ;
      NewDateTime = StartDateTime + ( 60 * 60 * 24 * 365.25 ) ;
      Put NewDate= Date9. ;
      Put NewDateTime = DateTime. ;
Run ;
```

```
NewDate=01JAN2011
NewDateTime=01JAN11:07:12:30
```

In the above example we see that in order to increase the Date value by one year we simply multiple the original value by 365.25 or the number of days on non leap years in the Gregorian calendar. This works until you have a leap year. To account for this you would have to know what year you are starting with, what year you want to end up with, and if you cross a leap year in the middle. A little later we will see how to overcome this problem with an easier solution.

The DateTime variable is increased with some more extensive math. Since DateTime values are in seconds we need to calculate the number of second in each minute for each hour for each day for the number of years by which we want to increase the original value. Both of these examples show that manually calculating dates/time values is less than efficient and can lead to errors or inaccuracies. Instead, SAS provides functions to simplify these calculations while also making them more precise.

DatDif Function
This function calculates the number of days between two dates. It has three arguments: DatDiff(StartDate , Ending Date, Basis). StartDate is the date from which you want to start counting. EndDate is the date at which SAS should stop counting. Basis tells SAS how to calculate the difference; it has two valid values '30/360' and 'ACT/ACT'.

'30/360' specifies a 30 day month and a 360 day year. Each month is considered to have 30 days, and each year 360 days, regardless of the actual number of days in each month or year whereas 'ACT/ACT' uses the actual number of days between dates. Here is an example showing the difference between the two:

```
Data _Null_ ;
      Date1 = '01Jan2010'd ;
      Date2 = '16Mar2010'd ;
      Diff1 = DatDif( Date1 , Date2 , '30/360'  ) ;
      Diff2 = DatDif( Date1 , Date2 , 'ACT/ACT' ) ;
      Put Diff1= Diff2= ;
Run ;

Diff1=75 Diff2=74
```

YrDiff Function
The YrDiff function is similar to the DatDif function except it calculates the number of years between two dates using a floating point number. Since it uses a floating point number it can handle more values for its Basis argument. The StartDate and EndDate arguments are the same, but there are more valid values for Basis.

'30/360' specifies a 30-day month and a 360-day year in calculating the number of years. Each month is considered to have 30 days, and each year 360 days, regardless of the actual number of days in each month or year. 'ACT/ACT' uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days that fall in 365-day years divided by 365 plus the number of days that fall in 366-day years divided by 366. 'ACT/360' uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 360, regardless of the actual number of days in each year. 'ACT/365' uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 365, regardless of the actual number of days in each year.

```
Data _Null_ ;
      Date1 = '01Jan2010'd ;
      Date2 = '16Mar2014'd ;

      Diff1 = YrDif( Date1 , Date2 , '30/360'  ) ;
      Diff2 = YrDif( Date1 , Date2 , 'ACT/ACT' ) ;
      Diff3 = YrDif( Date1 , Date2 , 'ACT/360' ) ;
      Diff4 = YRDif( Date1 , Date2 , 'ACT/365' ) ;

      Put Diff1= Diff2= Diff3= Diff4= ;
```

```
Run ;

Diff1=4.2083333333 Diff2=4.202739726 Diff3=4.2638888889 Diff4=4.2054794521
```

As we can see from the above output in the log YrDiff does use floating point numbers, producing partial years as its results, with variation depending on the Basis used. This can create a problem when trying to calculate how old someone is. If the above code had been used to determine someone's age, it would be confusing because we think of that value as a whole number yet all of these values are slightly more than four. While use of the CEIL or FLOOR function could help with the calculation, it would still be necessary to see whether the month and year of birth are before or after the month and year of the other date being used. So YrDiff, though useful, is not a one-step solution to calculating age.

In addition, YrDiff returns unexpected results when both the StartDate and EndDate fall in a leap year. Since the function uses a standard algorithm, this is not going to change, but does need to be taken into account.

Intck, Intnx, and Interval Types

The Intck and Intnx functions are two of the most useful Date/Time functions in SAS. Though they have their place, DatDif only counts the number of days between two dates and YrDif the number of years between two dates. Their limitations do not allow us to make more complex comparisons or calculations without doing extra math or having to know a great deal of detail about the data involved. Even well-documented code can lead to errors in these situations. To add to these deficiencies neither the DatDif nor YrDif function can adjust Date/Time values.

Intck and Intnx does not have these limitations and were specifically created to facilitate advanced Date/Time calculations. The Intck function counts intervals while Intnx adjust date/time values. They have the ability to utilize many different types of Date/Time intervals (single-unit, multi-unit, and shifted), frees the programmer from the constraints of only day or year intervals as well as eliminating additional manipulations when calculating Date/Time values. As we will see in the shift operators section, these intervals also can be manipulated to calculate aggregate intervals as one along with the ability to start the interval at different predefined points.

Intck Function

The Intck function calculates the number of times the beginning of a specified interval is crossed. Look closely at that sentence as it bears repeating along with an explanation concerning "beginning of a specified interval". This means that if your starting date falls at somewhere within the first interval you want to count, the Intck function would not count that interval as the first interval. The beginning of the specified interval units would not have been crossed until the next advancement of the interval between the two dates is crossed. To help us understand this let's look at a simple example.

Suppose you wanted to see how many months are in a year:

```
Data _Null_ ;
       StartDate = '01Jan2001'd ;
       EndDate   = '31Dec2001'd ;
              Months = Intck( 'Month' , StartDate , EndDate ) ;
       Put Months= ;
Run ;

Months=11
```

We have the exact starting date of the year along with the exact ending date. Now logically one would think that only the Intck function was needed with the month interval specified and it would give you the number of months between the two dates. However, as we see form the log we only get a value of 11 months. Since the starting date falls on the beginning of the specified interval, the beginning of the interval boundary is not crossed until you reach the 1st of February. Thus, that is where the Intck function starts its calculations. In order to achieve the correct result we need to either move the starting date back by one day or increase the ending date value by one day.

```
Data _Null_ ;
       StartDate = '01Jan2001'd ;
```

```
       EndDate   = '01Jan2002'd ;
              Months = Intck( 'Month' , StartDate , EndDate ) ;
       Put Months= ;
Run ;
```

Months=12

Since this can be tricky, let's look at a more complex example.

```
Data _Null_ ;
       StartDate = '31Dec2011'd ;
       EndDate   = '01Jan2012'd ;
              Days = Intck( 'Day'  , StartDate , EndDate ) ;
              Weeks = Intck( 'Weeks' , StartDate , EndDate ) ;
              Months = Intck( 'Month' , StartDate , EndDate ) ;
              QTR = Intck( 'QTR' , StartDate , EndDate ) ;
              Year = Intck( 'Year' , StartDate  , EndDate ) ;
       Put    Days=
              Weeks=
              Months=
              QTR=
              Year= ;
Run ;
```

```
Days=1 Weeks=1 Months=1 QTR=1 Year=1
```

We see that no matter which interval we selected the answer always is 1. The two dates selected are special in that no matter which of the four major single-unit intervals you choose they all cross their perspective beginning boundary only once.

Intnx Function

The Intnx function returns a SAS Date/Time value adjusted forwards or backwards by some number of specified intervals. In addition to adjusting the Date/Time value forwards or backwards, it also allows the user to specify the alignment of that date to Beginning, Middle, Ending, or SameDay within the interval. The default is Beginning.

```
Data _Null_  ;
       Format Date1-Date5 Date9.  ;
       StartDate = '11Jan2010'd ;
              Date1 = Intnx( 'Month' , StartDate , 4  ) ;
              Date2 = Intnx( 'Month' , StartDate , 4 , 'B'  ) ;
              Date3 = Intnx( 'Month' , StartDate , 4 , 'M'  ) ;
              Date4 = Intnx( 'Month' , StartDate , 4 , 'E'  ) ;
              Date5 = Intnx( 'Month' , StartDate , 4 , 'S'  ) ;
       Put    Date1=
              Date2=
              Date3=
              Date4=
              Date5=
;
Run ;
```

```
Date1=01MAY2010 Date2=01MAY2010 Date3=16MAY2010 Date4=31MAY2010 Date5=11MAY2010
```

We are showing five examples of the same basic Intnx function call, increasing the initial date value by four months while only varying the alignment of the resulting SAS Date/Time value. Date1 and Date2 have the same resulting Date value due to the fact that the default alignment is B or the beginning of the interval. Date3 is set to align the resulting date value to the middle of the month; for our example that would be 16May2010. Since May has 31 days there is no one midpoint, so the function will round up and place the date value on the 16th rather than the 15th. Date4 is the last day of the month for the ending date value. Finally the Date5 shows an example of using the SameDay alignment.

Two useful features of this function are that you can find the beginning and ending day of Date/Time value by simply specifying for it not to advance or subtract and intervals and using the B or E alignment argument. So whether it's years, months, days in a week, or the last Friday in a month you can find it with ease.

Here is an example:

```
Data _Null_ ;
      Format BegDate EndDate Date9. ;
            StartDate = '11Jan2010'd ;
      BegDate = Intnx( 'Month' , StartDate , 0 , 'B' ) ;
      EndDate = Intnx( 'Month' , StartDate , 0 , 'E' ) ;
      Put BegDate=
          EndDate=
      ;
Run ;
```

```
BegDate=01JAN2010 EndDate=31JAN2010
```

Something else to note is what happens when the SameDay alignment is used and the month you end with does not have that same number of days as the month with which you start. An example, if you use the SameDay alignment and Month interval with a starting date of 31jan2011 and only move the date by one month, what will be the result since January has 31 days but February only has 28. The Intnx function will increment your Date/Time value by the specified number of intervals, then, if it cannot find the exact day it is looking for, it simply backs the date up by one day until it finds a valid Date value.

```
Data _Null_ ;
      StartDate = '31Jan2011'd ;
            NewDate = Intnx( 'Month' , StartDate , 1 , 'S' ) ;
      Put NewDate= Date9. ;
Run ;
```

```
NewDate=28FEB2011
```

As we can see here, SAS simply backed the resulting Date value to the 28th of February, the last valid SAS Date value.

Intervals
Up until this point all the intervals used in the examples have been what are called single-unit intervals. In reality there are three types of intervals corresponding to how many parts are used: single-unit, multi-unit, and shifted-unit. Single-unit intervals are what we like to call the base intervals; they are the name of a valid SAS interval. Multi-unit intervals are a single-unit interval with a multiplier added on to increase the time period shifted by the Intck or Intnx functions. Shifted-units are intervals where the starting boundary of the interval is moved from its default to a new value. Single and multi-unit intervals are measured from the beginning of the month. Regardless of the starting Date/Time value SAS always calculates the starting point from a known point. So if the starting value is 11Mar2010 and the interval is month, SAS assumes the starting value of the interval is 01Mar2010.

Valid Single-Unit Intervals:
Time: Second, Minute, Hour

Date: Day, Week (seven day week starting on Sunday), WeekDay <#W> (seven day week with Sunday as the weekend), TenDay (ten days equal 1 interval), SemiMonth (half-month intervals), Month, QTR (three month intervals starting with: Jan, Apr, Jul, Oct), SemiYear (Six Month Intervals), Year
DateTime: prefix 'DT' to any of the Date values

Multi-unit intervals are created by simply adding a positive integer within the quotes to the end of the interval name. The default is assumed to be 1. While SAS can calculate the beginning of any single-unit interval, multi-unit intervals pose a problem. For example, if the Month2 interval is used and the starting date is in April, SAS can determine that April 1st is the first of the month but does that correspond to the beginning, middle and end of the two month multi-unit? To overcome this SAS starts every multi-unit interval (except for multi-week intervals) from January 1, 1960. Multi-week intervals start on December 27, 1959; this is due to the fact that January 1, 1960 was a Friday and thus the starting point to calculate the interval needed to be moved back to the prior Sunday.

```
Data _Null_ ;
      StartDate = '12Jan2011'd ;
            NewDate = Intnx( 'Month2' , StartDate , 2 , 'S' ) ;
      Put NewDate= Date9. ;
Run ;

NewDate=12MAY2011
```

In the above example we increase the initial starting date by two 2-month time periods and ask for the date to be on the same day as the starting Date. We see the resulting value is the 12th of May. Our starting time period would be the 1st of January, then it would be moved forward by two interval boundaries to the1st of March, then again to the 1st of May. Lastly, SAS moves the date to the appropriate day to match the starting value, which is the 12th.

Shifted-Units
Shifted-units are the most interesting and useful. They allow the user to shift the starting boundary of any valid interval value by adding a dot '.' and a positive integer not exceeding the number of shift periods in an interval. So Year.13 is not valid because there are not 13 months in a year, but Year2.13 is valid because there are a total of 24 months in two years and 13 is smaller than 24. With multi-unit, the change is a "shift by" this number while for shifted-units it is a "shift to" this number.

Interval	Shift Period
Year	Month
SemiYear	Month
QTR	Month
Month	Month
SemiMonth	SemiMonth
TenDay	TenDay
WeekDay	Day
Week	Day
Day	Day
Hour	Hour
Minute	Minute
Second	Second

Finding the Federal Fiscal Year
The Federal government's fiscal year starts on the 1st of October each year and ends on September 30 of the following year. Initially one would be tempted to use the Year function; however that would yield the calendar year and not the correct fiscal year for any date after the 1st of October. If one uses the Intnx function and shifts the start of the year to October, the 9th month of the year, and then uses the Year function, that would yield the correct fiscal year.

In the example below the base interval of Year is used with the optional shift value set to the 9th month by specifying 'Year.9'. The number of units to shift is set to 0 because we do not wish for the returned Date value to exceed the fiscal year we are currently in. The alignment value is set to 'M', but it could also have been set to 'E'. It could not be

left with the default of 'B' or set to 'S' as this would yield a Date value whose year would not have been set far enough ahead to return the correct results.

```
Data _Null_ ;
      FY = Year( Intnx( 'Year.9' , '15Jul2010'd , 0 , 'M' ) ) ;
            Put FY= ;
      FY = Year( Intnx( 'Year.9' , '10Dec2010'd , 0 , 'M' ) ) ;
            Put FY= ;
Run ;
```

```
FY=2010
FY=2011
```

Informats and Formats

SAS stores DateTime values as either the number of days or seconds from January 1, 1960. While this is the most efficient way to store and deal with DateTime values on a computer level, it does lead to two major problems. First, other software does not use the same base date as SAS; for example the Unix operating system uses the base date of January 1, 1970. Second, since SAS Date/Time values are stored as numeric values and few people, if any, can just look at a SAS Date/Time value and tell you the exact date and time it represents, we need a way to present these values in a human readable form. To do this SAS has provided both a set of informats and formats. While the authors would love to cover each and every one of these, there are simply too many so this section will cover the basic facts of Date/Time informats and formats as well as how to use them.

Informats

Date/Time informats are a way to tell SAS to convert a set of alpha-numeric characters, either from an external file or user input, into a SAS Date/Time value. One can either use one of the SAS-supplied informats or, if that does not meet the need, one can be created. However, it should be noted that to create your own informat requires preprocessing the data and creating that informat from the raw text values. Informats can be applied to text in one of four ways: INFORMAT statement, ATTRIB statement, INPUT statement, or INPUT function.

To use an informat in a INFORMAT statement, you simply specify the variable you want to associate with the informat, followed by the appropriate informat designation, width and a dot.

```
Data One ;
      Informat Date Date9. ;
            Infile Cards ;
            Input ID Date ;
      Cards ;
      1 01Jan1985
      2 23Mar1976
      3 14Dec1960
      ;
Run ;
```

To associate an informat using the ATTRIB statement, you state ATTRIB, the variable with which you want the informat associated, specify the appropriate informat followed by the correct width and a dot.

```
Data One ;
      Attrib Date Informat=Date9. ;
            Infile Cards ;
            Input ID Date ;
      Cards ;
      1 01Jan1985
```

```
        2 23Mar1976
        3 14Dec1960
        ;
Run ;
```

The use of the INPUT Statement is one of the most recognized methods as most SAS programmers first learn about informats in this context. In an INPUT statement, follow the variable you want the informat associated with by an appropriate informat and width.

```
Data One ;
        Infile Cards ;
        Input ID
                Date Date9. ;
        Cards ;
        1 01Jan1985
        2 23Mar1976
        3 14Dec1960
        ;
Run ;
```

Input Function
An input function allows the programmer to take a text variable and apply an informat to it, where the resulting formatted value is assigned to a SAS Date/Time variable or SAS statement. If the informat to be used is held in another SAS variable then the InputN or InputC statement should be used.

Here is an example using the Input function.

```
Data _Null_ ;
        Text = '01Jan2010' ;
                Date = Input( Text , Date9. ) ;
        Put Text=
            Date=
        ;
Run ;
```

```
Text=01Jan2010 Date=18263
```

Here is an example of using the InputN function.

```
Data _Null_ ;
        Text = '01Jan2010' ;
                Format = 'Date9.' ;
                Date = Input( Text , Format ) ;
        Put Text=
            Date=
        ;
Run ;
```

```
Text=01Jan2010 Date=18263
```

When needing to specify the format at run time you need to use the InputN or Input C statement. Use InputN when you want the resulting value to be numeric and InputC when you want it to be character. The great thing about specifying the informat at run time is you can have two or more records that require different informats being read into the same SAS Date/Time variable.

```
Data One ;
       Infile Cards ;
              Input DateTX $9. Format $ ;
       Cards ;
       01Jan2010 Date9.
       01012010  MMDDYY8.
       ;
Run ;

Data Two ;
       Set One ;
              Date = InputN( DateTx , Format ) ;
       Put DateTx=
           Format=
           Date=
       ;
Run ;

DateTX=01Jan2010 Format=Date9. Date=18263
DateTX=01012010 Format=MMDDYY8. Date=18263
```

Now that we have covered the basics of informats, we need to note a special type of when dealing with Date/Time values, namely the AnyDt family of informats. These were created to handle incoming Date/Times values that may or may not be all entered in the same form, but all are values for one Date/Time variable.

AnyDtDtew. creates a SAS Date value, AnyDtDtm. creates a SAS DateTime value and AnyDtTme. creates a SAS Time value. All three combines the following informats: Date, DateTime, DDMMYY, Julian, MMDDYY, MONYY, TIME, YYMMDD, and YYQ. One problem with trying to read multiple date values is sometimes you come across ones such as: 01-12-2001. The question arises as to whether 01 is the month value and 12 the value of day or vice versa. To solve this we use the DateStyle system option. This option allows the programmer to tell SAS the order of the values in the incoming text. The valid values are: MDY, MYD, YMD, YDM, DMY, DYM, and the default is Local. So if 01 is your month value and 12 is your day value, you would set the DateStyle to MDY; if the reverse is true, you would set it to DMY.

Formats

We saw in the previous section how one goes about getting a human readable text value of a Date/Time value into a SAS Date/Time value. While holding these values as numeric is great for computer manipulation, it does not facilitate human interpretation. To that end SAS uses formats to display the numeric values in a human readable and recognizable form. Like with informats we can associate a format with a particular variable by using one of the following four options: FORMAT statement, ATTRIB Statement, PUT statement, or in PROC Datasets.

In many of the previous examples you have seen a format in the PUT statement because the SAS Date/Time value is an integer and we need a method by which to show the reader what date the integer held in SAS represented.

```
Data One ;
       Date = '0Jan2010'd ;
       Put Date= Date9. ;
Run ;

Date=01JAN2010
```

While the example above is simple and also works when debugging, it is awkward when you have to do this every time. So it is recommended that one use a FORMAT statement and permanently associate a format with a variable.

```
Data One ;
```

```
       Format Date Date9. ;
              Date = '0Jan2010'd ;
       Put Date= ;
Run ;
```

```
Date=01JAN2010
```

Now we have the format, Date9., permanently associated with the variable Date in the data set, One. So barring the programmer stripping the format from the variable, either permanently or temporarily, every time SAS attempts to print the value of Date it will be in the Date9. form. Even if the data you used does not have formats associated with its Date/Time variables, SAS can handle this through the use of FORMAT statements in procedures as well as the DATA step.

```
Data One ;
       Date = '0Jan2010'd ;
Run ;

Proc Print Data = One ;
       Format Date Date9. ;
Run ;
```

What is interesting to note, both in this example and in our first example is that by using a format in the PUT statement or in a procedure you are temporarily assigning a format to that variable. This has two implications. First, it means that you do not have to have a permanent format associated with a variable in order to see a human readable value. Second, it means that you can have a permanent format associated with a variable and if you need to see it another way you can do so without altering the permanent format. In other words, you can override the permanent format for just that PUT statement or procedure's output.

Yet another way to permanenttly associate a format with a variable is with the ATTRIB statement. Just like with the informat, it starts with the ATTRIB keyword followed by a valid variable name, then the keyword Format, an equals sign and finally a valid format name.

```
Data One ;
       Attrib Date Format= Date9. ;
       Date = '01Jan2010'd ;
Run ;
```

The last way to permanently associate a format with a variable in SAS is through the use of PROC DataSets. This is a great method if you already have a data set and merely need to associate a format with a variable since it only affects the header information of the data set and keeps you from having to reread the data through another data step. It is a more efficient use of your time and system resources.

```
Data One ;
       Date = '01Jan2010'd ;
Run ;

Proc DataSets Lib = Work MemType = Data ;
       Modify One ;
       Format Date Date9. ;
Run ;
Quit ;
```

In this example we see that the variable in data set One has no format associated with it. PROC DataSets opens the header of data set One in the Work library. It then associates the Date9. format with the variable, Date.

Now that we have seen how to either permanently or temporarily associate a format with a variable, how can we reverse that assignment? If it was a temporary association, such as in the PUT statement or FORMAT statement in a procedure we don't have to worry about. However, if there is already a permanent format association, simply use the ATTRIB statement, FORMAT Statement (in a data step), or PROC DataSet procedure and do exactly the same as you did for permanently associating a format with a variable except leave off the format name. This will permanently remove that format's association with the variable.

Just like with the informats we also have a handful of special formats that we would like to discuss. Many times what is needed as output is simply a small modification of a format's current output. We have a format that almost meets our needs in the way of output but the separators between date or time parts need to be something other than the default. SAS has conveniently added such a feature onto a handful of the most popular formats. The following are those formats: DDMMYYxw., MMDDYYxw., MMYYxw., YYMMxw., YYMMDDxw., YYQxw, and YYQRxw. where the X represents one of the 5 separators(C=colon (:), D = dash(-), N=no separator, P = period (.), and S = Slash(/)) and W is the width. The formats MMDDYYxw. DDMMYYxw., and YYMMDDxw. also can have B = blank as a separator.

Date/Time Directives

So far we have looked at informats to take alpha-numeric characters and turn them into SAS Date/Time values as well using formats to change SAS Date/Time values into human readable values. The question then arises what happens when SAS does not have an informat or format that matches your needs. Unfortunately if you are dealing with informats, there is no easy solution and those that do exist are well outside of the scope of this paper. However, SAS does give the programmer the option to create their own specialized Date/Time format, which we call Date/Time Directives. These new programmer-defined DateTime formats can be applied anywhere a SAS supplied format can be used.

In order to create a user-defined Date/Time format, create a picture format with PROC Format, use the special directives in the formatted value area of the code and specify the DataType as being either Date,Time, or DateTime. DataType corresponds to the type of data you will be formatting with the user-defined format. Thus, if you are formatting a Date variable you will want your DataType to be Date, not Time or DateTime.

Date/Time Directives

%a	The Local's (as determined by your system setting) abbreviated weekday name.
%A	The Local's full weekday name.
%b	Local's abbreviated month name.
%B	Local's full month name.
%d	Day of the month as a decimal number (1-31), with no leading zero.
%H	Hour (24 hour clock) as a decimal number (0-23), with no leading zero.
%I	Hour (12 hour clock) as a decimal number (1-12), with no leading zero.
%j	Day of the year as a decimal number (1-366), with no leading zero
%m	Month as a decimal number (1-12), with no leading zero.
%M	Month as a decimal number (0-59), with no leading zero.
%p	Local's equivalent of either AM or PM.
%S	Second as a decimal number (0-59), with no leading zero.
%U	Week number of the year (Sunday as the first of the week) as a decimal number (0,53), with no leading zero.
%w	Weekday as a decimal number (1=Sunday, 7= Saturday).
%y	Year without century as a decimal number (0-99), with no leading zero
%Y	Year with a century as decimal number.

There are two important things to remember when you use these directives. First, they need to be enclosed in single and not double quotes. If they are in double quotes the macro processor will try resolve the % in the value. Secondly, the following directives do not by default place leading zeros in their resulting values: %d, %H, %I, %j, %m, %M, %S, %U, and %y. If you want them to display leading zeros you will need to place a zero between the % and the alpha character. For example, %y will produce a 7 while %0y will produce 07.

Let's say you are required to specify not only the day, full month name, and year, but also what day of the week it is for a report. One might think the only way to accomplish this is to take the SAS Date value and start breaking it apart, using the weekday function etc. Instead, we are going to show you how to this in one easy step.

```
Proc Format ;
      Picture MyDate
            . - .z    = 'Missing'
```

```
              Low - High = '%0d %B, %Y is on a %A'
              (DataType = Date) ;
Run ;



Data _Null_ ;
       Date = '12Jun2009'd ;
       Put Date= MyDate40. ;
Run ;

Date=12 June, 2009 is on a Friday
```

In the code above we create a Picture format call MyDate. which accounts for the possibility of missing SAS Date/Time values. The reason we did this is because if you do not code for missing values and SAS encounters one it will put an ERROR value instead of skipping it. This is followed by the actual Date directives describing the way in which we wish the values to be displayed by our format. First we have '%0d' which states we want the day value displayed with a leading zero if it is below 10. '%B' is asking for the full month name to be displayed next. The following '%Y' asks for the full four digit year value and finally the %A asks to print the full day of the week on which the date falls. You will also notice that we have text characters between directives, 'is on a', So not only can you specify how you want SAS to display the Date/Time parts you can also add your own characters or words to the displayed value. The final specification we make is a DataType of Date since we are formatting a SAS Date value. Had it been a DateTime value we would have to have specified DateTime for our DataType.

Example#2:

```
Proc Format ;
       Picture MyDate
              . - .z   = 'Missing'
              Low - High = '%0d %B, %Y is on a %A'
              (DataType = DateTime) ;
Run ;

Data _Null_ ;
       Date = '12Jun2009:12:03:30'dt  ;
       Put Date= MyDate40. ;
Run ;

Date=12 June, 2009 is on a Friday
```

Here we have essentially the same example as before except it is a SAS DateTime value we want to format. Remember from our initial description we wanted the date portion written out a certain way. In the above example all we need to do differently is substitute the DataType of DateTime instead of Date. SAS builds the format in essentially the same way with the exception that for the format to work it will require a SAS DateTime value. Since the directives only specify to write out the date parts of the DateTime value, the time value information is ignored when generating the formatted values.

Dates and Times in Macros

Programmers who are new to the macro language often find handling Date, DateTime, and Time values perplexing. The reason for this is that the macro language is essentially a text handling language, while the most efficient way to handle Date and Time values is to convert them to SAS Date and Time values, numeric scalars. To compensate for a lack of macro knowledge, programmers tend to use the more familiar `DATA _Null_` step to handle all the date and time manipulation.

The macro facility has two date variables that can be used, SysDate and SysDate9. These variables contain the date value from when the SAS session was started.

```
%Put Date = &SysDate ;

Date = 24JUN10

%Put Date = &SysDate9 ;

Date = 24JUN2010
```

Macro Example 1

```
%let Date = 01JAN2006 ;

Data _NULL_ ;
 Call Symput( 'Date' , put( input( "&Date" , Date9. ) , WordDate20. ) ) ;
Run ;

%put &Date ;

January 1, 2006
```

However, handling Date and Time values in a DATA _Null_ step unnecessarily complicates the macro code while preventing the programmer from creating a functional style of macro programming. We will cover how to use %Sysevalf and %Sysfunc along with Inputn, Putn, Intck, and Intnx to manipulate SAS Date and Time values in the macro language without the help of a DATA _Null_ step.

%SYSEVALF
Date and time literals are the easiest way to get a date and/or time value into a SAS Date and/or Time value so we shall start with these.

Macro Example 2

Data Step Code

```
Data _NULL_ ;
       Date = '01JAN2006'd ;
       DateTime = '01JAN2006:12:30:00'dt ;
       Time = '12:30:00't ;

       Put Date=
           Datetime=
           Time=  ;
Run ;

Date=16802 DateTime=1451737800 Time=45000
```

Macro Equivalent

```
%put Date = %Sysevalf( '01JAN2006'd )
     DateTime = %Sysevalf( '01JAN2006:12:30:00'dt )
     Time = %Sysevalf( '12:30:00't ) ;

Date = 16802  DateTime = 1451737800  Time = 45000
```

The date and time literal code is the same whether it is in the DATA step or the macro. The only difference is that to accomplish the same task in the macro language one has to use the literal values inside of the %Sysevalf function. While the %Eval function is used for most of the numeric operations in the macro language, it also performs character

comparisons. Thus the Date, DateTime, and Time that follow the quoted date and time values for literals are considered as true characters and not as part of the date and time values.

%SYSFUNC, INPUTN and PUTN

While date and time literals are straightforward, they allow for little flexibility in reading date and time values of different forms into SAS and they do not allow the programmer to format these SAS Date and Time values. To do this in the macro language one needs to move beyond the %Sysevalf function. In the DATA step these functions are performed by the INPUT and PUT functions and while the macro language can utilize most of the DATA step functions within the %Sysfunc function, the INPUT and PUT functions are not among these. Instead, %Sysfunc can use the InputN and PutN DATA step functions.

Macro Example 3

Data Step Code

```
Data _NULL_ ;
       Date = Input( '01JAN2006' , Date9. ) ;
       DateTime = Input( '01JAN2006:12:30:00' , DateTime18. ) ;
       Time = Input( '12:30:00' , Time8. ) ;
       Date2 = Put( Date , Date9. ) ;
       DateTime2 = Put( DateTime , DateTime18. ) ;
       Time2 = Put( Time , Time8. ) ;
              Put Date=
                  Date2=
                  Datetime=
                  DateTime2=
                  Time=
                 Time2=
              ;
Run ;
```

```
Date=16802          Date2=01JAN2006
DateTime=1451737800 DateTime2=01JAN06:12:30:00
Time=45000          Time2=12:30:00
```

Macro Equivalent

```
%let Date = %Sysfunc( InputN( 01JAN2006 , Date9 ) ) ;
%let Date2 = %Sysfunc( PutN(    &Date   , Date9 ) ) ;

%let DateTime = %Sysfunc( InputN( 01JAN2006:12:30:00 , DateTime18 ) ) ;
%let DateTime2 = %Sysfunc(   PutN(          &DateTime , DateTime18 ) ) ;

%let Time = %Sysfunc( InputN( 12:30:00 , Time8 ) ) ;
%let Time2 = %Sysfunc(   PutN(    &Time , Time8 ) ) ;

%put Date = &Date
     Date2 = &Date2
     DateTime= &DateTime
     DateTime2 = &DateTime2
     Time = &Time
     Time2 = &Time2 ;
```

```
Date         = 16802                    Date2        = 01JAN2006
DateTime     = 1451737800               DateTime2    = 01JAN06:12:30:00
Time         = 45000                    Time2        = 12:30:00
```

Please note that in the macro code the first argument to the InputN and PutN functions need not be quoted and doing so could introduce errors. Remember that the macro facility considers everything not preceded by a % or & to be text, which is different from the DATA Step where words have meaning and we quote the words we want to be considered as text. The other consideration is that the informat and format used in the InputN and PutN can be any valid SAS informat or format , but are specified without the trailing decimal. Here the macro facility uses decimals to determine the end of a macro variable's name. When a format in this context uses a decimal SAS has to make a decision as to whether the decimal means the end of a macro variable name or a format and would be consumed by the macro processor.

It should also be noted that %Sysfunc has a second argument which allows the programmer to add a format to be applied to the result of the function referenced by %Sysfunc.

Macro Example 4

```
%put %Sysfunc ( InputN( 01JAN2006 , Date9 ) , MonYY7 ) ;

JAN2006
```

INTCK AND INTNX

One of the more common problems that programmers encounter is finding how many units (days, months, years) one date is from another or how to increase or decrease a Date and/or Time value by a certain amount of units. Intck and Intnx are used in the DATA Step and have been previously discussed in the paper. Now, we will also use %Sysfunc in combination with these functions.

Macro Example 5

Data Step Code

```
Data _NULL_ ;
       StartDate = '01JAN2004'd ;
       EndDate = '01JAN2006'd ;
       NumOfMonths = Intck( 'Month' , StartDate , EndDate ) ;
       NewDate = Intnx( 'Month' , StartDate , NumOfMonths ) ;

       Put StartDate=
           EndDate=
           NumOfMonths=
           NewDate= Date9.
    ;
Run ;

StartDate= 16071    EndDate= 16802             NumOfMonths= 24    NewDate= 01JAN2006
```

Macro Equivalent

```
%let StartDate = %Sysevalf( '01JAN2004'd ) ;
%let EndDate = %Sysevalf( '01JAN2006'd ) ;
%let NumOfMonths = %Sysfunc( Intck( Month , &StartDate , &EndDate ) ) ;
%let NewDate = %Sysfunc( Intnx( Month , &StartDate , &NumOfMonths ) ) ;

%put StartDate = &StartDate
     EndDate = &EndDate
     NumOfMonths = &NumOfMonths
     NewDate  = &NewDate ;

StartDate = 16071      EndDate   = 16802      NumOfMonths = 24      NewDate = 16802
```

The constant text values of the Intck and Intnx functions do not need to be in quotes. Otherwise they work in exactly the same as they do in the DATA Step. Finally, notice the value of the Macro variable NewDate; it is a SAS Date value. While this is good for manipulating the value, it is hard for humans to read. The simplest way to change the printed values is to use the second argument to the %Sysfunc as in the example below.

Macro Example 6

```
%let NewDate = %Sysfunc( Intnx( Month , &StartDate , &NumOfMonths ) , WordDate20 ) ;
%put NewDate = &NewDate ;

NewDate = January 1, 2006
```

Conclusion

Though working with SAS Date, Time and DateTime values can seem confusing, this paper covers the history behind their structure, how to manipulate them and factors to consider when needing to output their values. By knowing the logic inherent in SAS Date/Time values, a programmer can more easily bring them in from external data, manipulate them within existing data or produce them in reports in almost any desired configuration. From informats to macros, incrementing to converting, we have presented a comprehensive discussion of these values and hopefully simplified the process of working with them. SAS may have your number, but that doesn't mean you can't turn it into something recognizable by anyone with a clock or a calendar!

References

TS-DOC: TS-668 - SAS Dates, Times, and Interval Functions

Dunn, Toby "Handling Dates in the Macro Facility"
analytics.ncsu.edu/sesug/2006/SC11_06.PDF

Chakravarthy, Venky "Have a Strange DATE? Create your own INFORMAT to Deal with Her"
www2.sas.com/proceedings/sugi27/p101-27.pdf

Morgan, Derek "The Essential Guide to SAS Dates and Times"
SAS Press 2006

SAS OnlineDoc 9.1.3 for the Web
SAS Institute Inc., Cary, NC

Contact Information

Your comments and questions are valued and encouraged. Contact the authors at:

Sarah Woodruff
Westat
1600 Research Boulevard
WB 401
Rockville, MD 20850
(240) 314-7562
SarahWoodruff@Westat.com

Toby Dunn
AMEDDC&S (CASS)
San Antonio, TX
Toby.Dunn@amedd.army.mil