Does Anyone Know What Time (or Day) It Is?

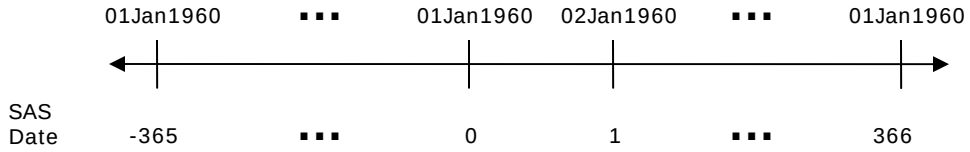
Keith Cranford
Office of the Attorney General of Texas, Child Support Division

Abstract

SAS date and time functions provide flexibility to do many things. This paper will provide a comprehensive overview of how SAS dates, times and datetimes work. Many of the SAS date/time functions, formats and informats will also be presented. Many will be familiar, but some are either new to SAS9 or contain new features.

Working with Dates

A SAS date value is simply the number of days between January 1, 1960 and the date being stored. For example, January 1, 1960 is stored as 0 (zero), January 2 as 1, and January 1, 1961 as 366 (1960 was a leap year), etc.



Storing dates as numeric values allows you to use arithmetic operators with dates. For example, you can find how many days have elapsed between dates by subtracting the two dates (date1 – date2), or a common way to compute someone's current age is to determine a person's age in days and divide by the approximate number of days in a year (int((today()-birth_date)/365.25)).

However, this is only scratching the surface. SAS provides a host of functions, formats, and informats to use when working with dates. These allow you to create date values, extract parts of date values, and work with various date intervals. The next sections will cover these useful tools.

Creating Dates

SAS provides two special functions and a general function that can be used to create SAS dates. The MDY and TODAY functions are functions that are specifically designed to create SAS dates. The syntax for these functions is as follows.

MDY(*month,day,year*)
TODAY()

Both of these return SAS date values. The MDY function returns the date value associated with the month, day, and year specified, and the TODAY function returns the date value associated with the current date. You can also use DATE() as an alternative to TODAY(). Note that the TODAY function has no argument, but the parentheses are necessary to distinguish it as a function.

The INPUT function can also be used to create a SAS date value by converting a character string containing a date in some format into a SAS date. The general form follows.

INPUT(*source*, *informat*.)

The source could be a character variable containing a formatted date or a character constant. The informat will be the appropriate numeric date informat to read the source. The result will be the date value associated with the source. More information will be provided on date informats in a subsequent section.

To illustrate these functions consider the following data step code.

```
**-- Creating Dates in SAS --**;
data date_create ;
      var1 = mdy(10,2,2007) ;
      var2 = today() ;
      var3 = input('10/2/2007',mmddyy10.) ;
      var4 = input('02OCT07',date7.) ;
      var5 = input('10/2/2007',anydtdte.) ;
      var6 = input('02OCT07',anydtdte.) ;
      format var1-var6 mmddyy10. ;
run ;
```

A print of this data produces the following output.

Creating Dates in SAS

Obs	var1	var2	var3	var4	var5	var6
1	10/02/2007	07/31/2007	10/02/2007	10/02/2007	10/02/2007	10/02/2007

The MDY function returns the value 17441 for the variable var1, which has been formatted to display 10/02/2007. This code was run on July 31, 2007, so the TODAY function created a date for that day. The INPUT function created SAS date values for Oct 2, 2007, by converting the character strings based on the proper informats. Informats will be discussed in more detail later in the paper, but notice the ANYDTDTE informat. It reads a variety of dates, which can be handy if dates have been input in a number of different ways in the same field.

Extract Part of a Date

SAS functions can also be used to extract a part or characteristic of a date. For example, you may need the year or the month of a date, or need the quarter or weekday associated with a date. The following table summarizes functions that extract a part of a date.

Function	Description	Syntax
DAY	Returns the day of the month from a SAS date value	**DAY**(*date*)
MONTH	Returns the month from a SAS date value	**MONTH**(*date*)
YEAR	Returns the year from a SAS date value	**YEAR**(*date*)

QTR	Returns the quarter of the year from a SAS date value	**QTR**(*date*)
WEEK	Returns the week-number value	**WEEK**(<*sas_date*>, *descriptor*>)
WEEKDAY	Returns the day of the week from a SAS date value	**WEEKDAY**(*date*)

The form of these functions is similar, except for the additional parameter for WEEK. They all simply take a SAS date value and return the appropriate requested value.

The following example illustrates these functions.

```
data date_extract ;
      date = '02Oct2007'd ;
      day = day(date) ;
      weeku = week(date,'U') ;
      weekv = week(date,'V') ;
      weekw = week(date,'W') ;
      weekday = weekday(date) ;
      quarter = qtr(date) ;
      month = month(date) ;
      year = year(date) ;
      format date mmddyy10. ;
run ;
```

The result is…

Extracting Parts of Dates in SAS

Obs	date	day	weeku	weekv	weekw	weekday	quarter	month	year
1	10/02/2007	2	39	40	40	3	4	10	2007

The DAY function returns 2 (second day of the month), WEEKDAY returns 3 (Oct 2 is a Tuesday, the 3rd day of the week), QTR function returns 4 (October is in the 4th quarter of the year), MONTH returns 10 (October is the 10th month), and YEAR returns 2007 (it's 2007).

The WEEK function returns different values depending on the descriptor parameter. From the SAS documentation,

> ***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 and uses a leading zero and a maximum value of 53.
>
> ***V*** specifies the SAS date value. The number-of-week value is represented as a decimal number in the range 01-53 and uses a leading zero and 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 and uses a leading zero and a maximum value of 53.

With U, October 2 is in the 39th week of 2007, if the first week of the year is considered week 0 and weeks start on Sunday. With V, October 2 is in the 40th week, since the first week is considered week 1 and January 4th and the first Thursday are contained in that week. Starting the week on Monday does not have an effect, since Tuesday would occur in the same week regardless. Using W is very similar to V, except if January 1 falls on a Sunday. In this case, January 1 would be week 0 for W, but would be 53 for V. This happened in 2006.

Interval Functions

An interesting set of functions is those that work with date intervals, INTCK and INTNX. The syntax and description of these functions are given below.

Function	Description	Syntax
INTCK	Returns the integer count of the number of interval boundaries between two dates, two times, or two datetime values	**INTCK**(*interval*, *from*, *to*)
INTNX	Increments a date, time, or datetime value by a given interval or intervals, and returns a date, time, or datetime value	**INTNX**(*interval*<*multiple*><*.shift-index*>, *start-from*, *increment*<,*alignment*>)

The major difference in the two functions is what they return. INTCK returns the number of interval boundaries crossed, whereas the INTNX returns a date (or time or datetime) value. The INTCK function requires two dates, a start and an end date. The function then counts the number of interval boundaries crossed as you move from the start date to the end date. INTNX requires one date, a start date. The increment argument specifies how many intervals to move from this start date. The alignment specifies where within the interval to assign the date. You can specify b (beginning – the default), m (middle), e (end), or same (same as start date).

Each of the functions works with intervals. The simple intervals are DAY, WEEK, WEEKDAY, MONTH, QTR, and YEAR. These work in a fairly straightforward manner. However, you can add a multiple and/or a shift index to produce a more complicated interval. The multiple will multiply the interval. For example, DAY3 will use a 3-day interval. The shift index will shift the interval. The interval of the shift depends on the initial interval. For example, YEAR.9 will shift the year interval to start on the 9th month resulting in the year starting in September, or MONTH2.2 will shift the 2-month interval to the second month resulting in Feb-Mar, Apr-May, etc. As you can see this adds a lot of flexibility, but can be a bit tricky as well. There are two good sources that discuss this further, Chapter 4 of Ron Cody's SAS Functions by Example and a technical note, TS-668, "SAS Dates, Times, and Interval Functions", which can be found at http://ftp.sas.com/techsup/download/technote/ts668.html.

The following example illustrates the INTCK function and some its uses.

```
**-- INTCK function --**;
data date_interval1 ;
      date1 = '01jan06'd ;
```

```
            date2 = '31dec06'd ;
            date3 = '01jan07'd ;

            yrs1_2 = intck('year',date1,date2) ;
            yrs1_3 = intck('year',date1,date3) ;
            yrs2_3 = intck('year',date2,date3) ;

            mos1_2 = intck('month',date1,date2) ;
            mos1_3 = intck('month',date1,date3) ;
            mos2_3 = intck('month',date2,date3) ;

            format date1-date3 mmddyy10. ;
      run ;
```

The resulting data set is shown below.

INTCK Function

Obs	date1	date2	date3	yrs1_2	yrs1_3	yrs2_3	mos1_2	mos1_3	mos2_3
1	01/01/2006	12/31/2006	01/01/2007	0	1	1	11	12	1

The INTCK function counts the time a boundary is crossed, where the boundaries start at January 1, 1960. Note that no year boundary is crossed from January 1 to December 31, so yrs1_2 equals zero. However, one year boundary is crossed from December 31, 2006 to January 1, 2007, so yrs2_3 equals one. (Think of it as how many times you celebrated at a New Year's Eve party between the given dates!) These are somewhat extreme examples, but they show how the function works. A more typical example, from Jan 1, 2006 to Jan 1, 2007 results in value of one year boundary crossed. The month interval works similarly.

The next example illustrates the simpler uses of INTNX.

```
      data date_interval2 ;
            date_start = '02Oct07'd ;

            date_1month = intnx('month',date_start,1) ;
            date_1year = intnx('year',date_start,1) ;
            date_1qtr = intnx('qtr',date_start,1) ;
            date_12mos = intnx('month',date_start,12) ;
            date_monthsame = intnx('month',date_start,12,'same') ;
            date_monthend = intnx('month',date_start,0,'e') ;

            format _all_ mmddyy10.;
      run ;
```

The resulting data set is shown below.

INTNX Function

Obs	date_start	date_1month	date_1year	date_1qtr	date_12mos	date_monthsame	date_monthend
1	10/02/2007	11/01/2007	01/01/2008	01/01/2008	10/01/2008	10/02/2008	10/31/2007

Moving the start date one month ahead results in November 1, 2007, the first day of the next month. Moving the start date one year ahead results in January 1, 2008, the first day of the next year, and one quarter ahead gives the same date, since this is also the first day of the next quarter. Again, note that these are based on intervals starting on January 1, 1960. Advancing the date 12 months ahead results in October 1, 2008. This would probably be a better way to advance the date a year, especially using the alignment argument of "same," which is what is done in the date_monthsame variable. Finally, you can move a date to the end of the month by using the e alignment argument and a zero increment.

The following example shows some of the effects of using multiplier and shift indices,

```
data date_interval3 ;
      date_start = '02Oct07'd ;

      date_1fyear = intnx('year.9',date_start,1) ;
      date_currfy = intnx('year.9',date_start,0) ;
      date_currqtr = intnx('qtr.3',date_start,0,'e') ;
      currqtr = qtr(intnx('month',date_start,4,'b')) ;

      date_month3 = intnx('month3.3',date_start,0,'b') ;

      format _all_ mmddyy10.;
      format currqtr 2.;
run ;
```

with the resulting data set.

INTNX Function - Multiplier/Shift

Obs	date_start	date_ 1fyear	date_ currfy	date_ currqtr	currqtr	date_ month3
1	10/02/2007	09/01/2008	09/01/2007	11/30/2007	1	09/01/2007

This example basically illustrates ways to work with a fiscal year that begins on September 1. The YEAR.9 interval indicates a year interval that begins in the 9th month (September 1). The first variable, date_1fyear, increments a date to the beginning of the next fiscal year, and the second, date_currfy, moves the date to the beginning of the current fiscal year. A similar technique can be used for quarters by using QTR.3, which indicates to begin the quarters in the 3rd month (and, hence, the 6th, 9th and 12th months). Designating the quarter numbers correctly requires just a little trick of advancing the date 4 months (e.g., moving September 1 to January 1) and applying the QTR function on the new date. Finally, the last variable shows the use of both the multiple and shift index. MONTH3.3 is equivalent to QTR.3, since it will create three month intervals, starting in the 3rd month.

Working with the multiple and shift index can be a little tricky and may require some trial and error to get exactly what you want. However, the result is an easy way to set up interval boundaries that meet your business requirements and are, then, easy to apply.

Date Formats and Informats

SAS provides many formats to display date values and quite a few informats to read and create SAS date values. These include variations on the month/day/year theme, spelling out the months, including day of week, displaying quarter or week data.

The following table shows various date formats with the default display for October 2, 2007.

Format	Display	Format	Display
DATE	02OCT07	WEEKDATX	Tuesday, 2 October 2007
DAY	2	WEEKDAY	3
DDMMYY	02/10/07	WEEKU	2007-W39-03
DDMMYYD	02-10-07	WEEKV	2007-W40-02
DOWNAME	Tuesday	WEEKW	2007-W40-02
JULDAY	275	WORDDATE	October 2, 2007
JULIAN	07275	WORDDATX	2 October 2007
MMDDYY	10/02/07	YEAR	2007
MMDDYYD	10-02-07	YYMM	2007M10
MMYY	10M2007	YYMMS	2007/10
MMYYS	10/2007	YYMMDD	07-10-02
MONNAME	October	YYMMDDS	07/10/02
MONTH	10	YYMON	2007OCT
MONYY	OCT07	YYQ	2007Q4
QTR	4	YYQS	2007/4
QTRR	IV	YYQR	2007QIV
WEEKDATE	Tuesday, October 2, 2007	YYQRD	2007-IV

Most of these are self-explanatory, but a few need further clarification. Those formats that have a D or S at the end, such as DDMMYYD, have the option to indicate what separator to use.

B separates with a blank

C separates with a colon

D separates with a dash

N indicates no separator

P separates with a period

S separates with a slash

Also, note the WEEK formats correspond to the WEEK function discussed above. The first part is the year, the second the week within the year, and the last is the day within the week. The weeks match what was shown earlier, and the day of week for WEEKU starts on Sunday while WEEKV and WEEKW start on Monday.

Creating SAS date values from date informats work similarly to the date formats. In fact, except for one informat, there are corresponding formats for each informat. The table below lists the date informats available with an example source and the resulting SAS date value (formatted).

Informat	Source	Result
ANYDTDTE	10/02/07	10/02/2007
DATE	02OCT07	10/02/2007
DDMMYY	021007	10/02/2007
JULIAN	07275	10/02/2007
MMDDYY	100207	10/02/2007
MONYY	1007	10/01/2007
WEEKU	07W3903	10/02/2007
WEEKV	07W4002	10/02/2007
WEEKW	07W4002	10/02/2007
YYMMDD	071002	10/02/2007
YYMMN	0710	10/01/2007
YYQ	07Q4	10/01/2007

These all produce a SAS date value corresponding to October 2, 2007, except for the three that don't have day inputs. MONYY and YYMMN give in the first day of the month, and YYQ gives the first day of the quarter. The WEEK informats would also give the first day of the week, if a day is not specified.

Working with Time and Datetime

SAS handles time and datetime values in a similar manner as dates. Time is stored as the number of seconds since midnight, and datetime is stored as the number of seconds since midnight, January 1, 1960. SAS then supplies various functions for creating time or datetime values and displaying these values.

Creating Time and Datetime Values

SAS has functions to create time and datetime values. The HMS and TIME functions create time values, while the DHMS and DATETIME functions create datetime values. Also, the INPUT function, using the proper informat, can create date or datetime values. The syntax for these functions follow.

HMS(*hour,minute,second*)
TIME()
DHMS(*date,hour,minute,second*)
DATETIME()

In the HMS and DHMS functions, the *hour*, *minute* and *second* arguments are numeric, and the *date* must be a date value (could be a date constant). The TIME and DATETIME functions do not have any arguments.

The INPUT function can produce time or datetime values similar to what you saw earlier with date values. The syntax is the same as discussed above. The only difference is you must use date or datetime informats, such as TIME or DATETIME.

The following example illustrates the use of these functions.

```
**-- Creating Times and DateTimes in SAS --**;
data time_create ;
      var1 = hms(16,0,0) ;
```

```
        var2 = time() ;
        var3 = input('16:00:00',time.) ;
        var4 = dhms('02Oct07'd,16,0,0) ;
        var5 = datetime() ;
        var6 = input('02Oct07:16:00:00',datetime.) ;
        format var1-var3 time8. var4-var6 datetime. ;
run ;
```

A print of the resulting data set is shown below.

```
                    Creating Times in SAS

 Obs        var1        var2        var3          var4

  1      16:00:00    13:40:51    16:00:00    02OCT07:16:00:00

 Obs          var5                 var6

  1      07AUG07:13:40:51    02OCT07:16:00:00
```

The HMS function and the INPUT function using TIME, var1 and var3, return the time value for 4:00 p.m. The actual value stored is 57600, which is formatted here for convenience. The DHMS and the INPUT function using DATETIME, var4 and var6, return the date time value for October 2, 2007 at 4:00 p.m. The TIME and DATETIME functions return the time and datetime, respectively, of when the program was run. In this example, these are 1:40 p.m. and 1:40 p.m. on August 7, 2007.

Extracting Parts of Time and Datetime

SAS can extract parts of time and datetime values. You can extract the hour, minute and seconds from a time or datetime value, or you can extract the date part or the time part from a datetime value. With the date part result, you could subsequently extract the parts of date value. The following table gives each function's description and syntax.

Function	Description	Syntax
HOUR	Returns the hour from a SAS time or datetime value	**HOUR**(*time* | *datetime*)
MINUTE	Returns the minute from a SAS time or datetime value	**MINUTE**(*time* | *datetime*)
SECOND	Returns the second from a SAS time or datetime value	**SECOND**(*time* | *datetime*)
DATEPART	Extracts the date from a SAS datetime value	**DATEPART**(*datetime*)
TIMEPART	Extracts a time value from a SAS datetime value	**TIMEPART**(*datetime*)

The form of these functions is all the same, except that the first three can take either a date or a datetime while the last two requires a datetime value.

The following example illustrates these functions.

```
**-- Extracting parts of Times in SAS --**;
data time_extract ;
```

```
        dttime = '02Oct2007:16:00:00'dt ;
        time = '16:00:00't ;
        hour = hour(time) ;
        minute = minute(time) ;
        second = second(time) ;
        hour2 = hour(dttime) ;

        date = datepart(dttime) ;
        time2 = timepart(dttime) ;

        format time time2 time. ;
        format dttime datetime. ;
        format date mmddyy10. ;
    run ;
```

A print of the data set is shown below.

```
                    Extracting Parts of Times in SAS

 Obs      dttime        time    hour minute second hour2       date   time2

  1   02OCT07:16:00:00 16:00:00  16     0      0     16   10/02/2007 16:00:00
```

The HOUR, MINUTE and SECOND extracts these parts from the time value of 4:00 p.m. as 16 (military time), 0, and 0, respectively. Notice that HOUR can be applied to the datetime value (hour2) and give the hour part of the datetime. Finally, DATEPART and TIMEPART extract the date and time parts of the datetime variable.

Interval Functions, Again

The interval functions, INTCK and INTNX, can be used with date and datetime values as well as date values. The difference is which intervals you use. Some of the relevant intervals are SECOND, MINUTE, and HOUR that work with time and datetime values and DTDAY, DTWEEK, DTMONTH, and DTYEAR that work with datetime values. Other than this, these functions work the same way, including the use of multiples and shift indexes.

Time and Datetime Formats and Informats

There are a number of formats and informats that can be used with time and datetime values, but not as many as with dates. These work in much the same way as with dates.

The following table shows how the time value 4:04:37 p.m. or the datetime value 4:04:37 p.m. on October 2, 2007.

Format	Display	Format	Display
DATETIME	02OCT07:16:04:37	TOD	16:04:37
DATEAMPM	02OCT07:04:04:37 PM	HHMM	16:05
DTDATE	02OCT07	HOUR	16
DTMONYY	OCT2007	MMSS	964:37
DTWKDATX	Tuesday, 2 October 2007	TIME	16:04:37
DTYEAR	2007	TIMEAMPM	4:04:37 PM
DTYYQC	2007:4		

Most of these are straightforward. Notice, though, that the two AMPM formats display the time with AM or PM, TOD displays the time from a datetime value, and MMSS displays the minutes in a day from midnight to the time value (964 = 16*60 + 4).

The following table shows how you can read a source and create a time or datetime value.

Informat	Source	Result (as a date or datetime)
ANYDTDTM	10022007	02OCT07:00:00:00
ANYDTTME	02OCT07:16:04:37	16:04:37
DATETIME	02OCT07:16:04:37	02OCT07:16:04:37
TIME	16:04:37	16:04:37
STIMER	964:37	16:04:37

The ANYDTDTM informat creates a datetime value from a source using any of a list of informats such as DATE, DATETIME, MMDDYY, TIME, etc. The ANYDTTME informat does the same, except that it creates a time value. The DATETIME and TIME informats require the sources to be in the standard datetime and time format. Lastly, the STIMER informat allows a source of various time values, such as hh:mm:ss or mm:ss.

Conclusion

SAS handles dates and times very well. You have the ability to read, create, and display these values in just about any manner that you would like. The nice aspect is that once you have a variable stored as a date, time, or datetime value, you can display it many ways and can apply all of the SAS functions that apply to these values. This makes working with dates, times, and datetimes very flexible. Hopefully, you now know what time (or day) it is!

References

Cody, Ron. 2004. *SAS® Functions by Example.* Cary, NC: SAS Institute, Inc.

Contact Information

If you have questions, please contact the author at:
Keith.cranford@cs.oag.state.tx.us