Working with Dates and Times in Stata

This document introduces Stata's functions for working with dates and times, and addresses some common questions. This is intended as a pared-down, quick reference covering a subset of Stata's capabilities. The Stata Data Management Guide offers an excellent, more detailed reference covering additional topics; it can be found in the Velma Denning Room in Green Library. Stata's Help pages have similar coverage, available online here: http://www.stata.com/help.cgi?dates_and_times#how.

Table of Contents

Understanding How Stata Stores Dates and Times

Stata stores dates and times as integers, representing the number of units (e.g. hours, days, weeks, or years) since January 1, 1960. Negative numbers represent dates and times before January 1, 1960, and positive numbers represent dates and times afterward. The specific date format tells Stata whether to interpret the integer as a number of milliseconds, minutes, days, etc.

Here is how different date and time values are coded:

		Numerical value and interpretation		
Format	Meaning	Value = -1	**Value = 0**	Value = 1
		31dec1959	**01jan1960**	01jan1920
%tc	clock	23:59:59.999	**00:00:00.000**	00:00:00.001
%td	days	31dec1959	**01jan1960**	02jan1960
%tw	weeks	1959w52	**1960w1**	1960w2
%tm	months	1959m12	**1960m1**	1960m2
%tq	quarters	1959q4	**1960q1**	1960q2
%th	half years	1959h2	**1960h1**	1960h2
%tg	generic	-1	**0**	1

Source: Stata Help Pages

Explanation: The middle, bolded column shows the base value. For a %td value, 0 means 01jan1960. The table also shows that -1 means 31dec1959 and 1 means 02jan1960. A %td value records the number of days from 01jan1960; a %tc value records the number of milliseconds from the start of 01jan1960; a %tw value records the number of weeks from the first week of 1960; and so on.

You may have noticed that there is no variable type for years. Stata does in fact have one, %ty. Unlike other date and time variables, %ty variables start from 0 AD; years 100 through 9999 are valid:

		Numerical value and interpretation		
Format	Meaning	1959	1960	1961
%ty	year	1959	1960	1961

Source: Stata Help Pages

Finally, Stata also includes a %tC variable type, which is a clock variable set to account for leap seconds. To learn more about this, see the Data Management Guide or the Stata Help pages.

Recommended Storage Types for %t Variables

It is important to remember that **%tc and %tC variables must be stored as doubles (the largest numeric data type).**

%td variables may be stored as a *float* or *long* (a long is larger than a float, but smaller than a double).

All other **%t** variables may be stored as a *float* or *int* (the smallest numeric data types).

Entering Dates and Times into Stata

The most straightforward way to do this is to start by entering the dates as a string in a standard format, such as "June 16, 2010" or "6-6-2010". Then use a conversion function (see below) to convert the *string* to a %t variable.

Converting Strings to Dates and Times

To convert a *string* to a date and time variable, you first need to decide what type of date and time variable you want to end up with (%tc, %td, %tw, etc.) in order to choose the appropriate conversion function. Then, you need to specify the appropriate *mask*, based on the format of your date-strings. The *mask* tells Stata about the form of the input, and the conversion function tells Stata about the form of the output.

Here are the string-to-numeric conversion functions:

Format	String-to-numeric conversion function
%tc	**clock(***string***,** *mask***,** [**,** *baseyear*]**)**

%tC	**Clock(***string***,** *mask***,** [**,** *baseyear*]**)**
%td	**date(***string***,** *mask***,** [**,** *baseyear*]**)**
%tw	**weekly(***string***,** *mask***,** [**,** *baseyear*]**)**
%tm	**monthly(***string***,** *mask***,** [**,** *baseyear*]**)**
%tq	**quarterly(***string***,** *mask***,** [**,** *baseyear*]**)**
%th	**halfyearly(***string***,** *mask***,** [**,** *baseyear*]**)**
%ty	**yearly(***string***,** *mask***,** [**,** *baseyear*]**)**

Source: Stata Help Pages

string is the value to be translated. *baseyear* is an optional argument, and is described in the section "Working with Two-digit Years" below.

mask specifies the order of the components, using the following abbreviations:

Abbreviation	Value	Allowable Range
M	month	$1 \leq M \leq 12$ Jan $\leq M \leq$ Dec January $\leq M \leq$ December
D	day	$1 \leq D \leq 31$
Y	year	$100 \leq Y \leq 9999$
h	hour	$0 \leq h \leq 23$
m	minute	$0 \leq m \leq 59$
s	second	$0.000 \leq s \leq 59.999$
W	week	$1 \leq W \leq 52$
Q	quarter	$1 \leq Q \leq 4$
H	half year	$1 \leq H \leq 2$

For more information on formatting codes, which allow you to display months by number, abbreviation, or full name (among other formatting choices) see "Formatting date and time values" in the Data Management Guide or in the Help pages.

Here are some examples of date strings and their corresponding masks:

Date string	Mask
"August 21, 2005"	"MDY"
"8-21-2005"	"MDY"
"21 August, 2005"	"DMY"
"21-8-2005"	"DMY"
"8-21-2005 12:35"	"DMYhm"
"21aug2005 12:35:22"	"DMYhms"

Working with Two-digit Years

There are two options for working with two-digit years. First, if all years fall in the same century (i.e. all years start with the same two-digit prefix) you can specify the appropriate prefix in the

mask. For example, if you have the string "11-4-08", you can specify the mask "DM20Y", and Stata will read it as November 4, 2008. Or you can specify "DM19Y" or "DM18Y" to specify the date as November 4, 1908 or November 4, 1808, respectively. You can also add hours, minutes, and seconds onto these masks, such that "11-4-08 10:23:05" would be given the mask "DM20Yhms".

On the other hand, sometimes a dataset contains dates from more than one century. For example if your data covers 1990-2005, you would want "5-5-97" to be interpreted as being in 1997, but "8-10-03" to be interpreted as being in 2003. This is where *baseyear* is useful. When you specify *baseyear*, you tell Stata to choose a prefix that places the year closest to *baseyear*. For example, if your dates are stored in a variable called `timestring`, and you want to create a `%td` variable called `timestamp`, you could use this command:

```
generate double timestamp = date(timestring, "DMY", 2000)
```

Stata would then interpret the two digit year 97 as 1997, because 1997 is closer to 2000 than 2097 (or 1897, 1797, etc.) is. It would interpret 03 as 2003, because 2003 is closer to 2000 than 1903 is.

Converting Between %t Variable Types

Once you have a `%t` variable, it's easy to change it to any other type of `%t` variable. In general, you first have to convert it to `%td`, and then you can convert it to any other type. Here are the functions to use:

	To…							
From	%tc	%tC	%td	%tw	%tm	%tq	%th	%ty
%tc		Cofc()	dofc()					
%tC	cofC()		dofc()					
%td	cofd()	Cofd()		wofd()	mofd()	qofd()	hofd()	yofd()
%tw			dofw()					
%tm			dofm()					
%tq			dofq()					
%th			dofh()					
%ty			dofy()					

For example, to convert a `%td` value to a `%tc` value:

```
generate double datetimevalue = cofd(datevalue)
```

To convert a `%tq` value to a `%tc` value, you have to go through `%td`:

```
generate double datetimevalue = cofd(dofq(quartervalue))
```

Working with Incomplete Dates and Times

The raw string variable does not dictate which type of `%t` variable you can convert it to. For example, you could convert "5-5-97" to a `%tc` variable, which would just have 0's for its hour, minute, and second values.

See the Data Management Guide or the Help pages for more information.

Extracting Date and Time Components

You may only want to work with years, months or quarters even though you have complete dates. If you want to obtain date components (components the size of a day or longer) you first need to convert your variable to a `%td` variable.

Let *d* be a `%td` variable or value. These functions will extract components of *d*:

Function	Returns	Result if d = td(05jul1972) (i.e., d = 4,569)
`year`(*d*)	calendar year	1972
`month`(*d*)	calendar month	7
`day`(*d*)	day within month	5
`doy`(*d*)	day of year	187
`halfyear`(*d*)	half of year	2
`quarter`(*d*)	quarter	3
`week`(*d*)	week within year	27
`dow`(*d*)	day of week (0 = Sunday)	3 (= Wednesday)

If you want time components (of size less than 1 day), you need to start with a `%tc` or `%tC` variable. Let *t* be a `%tc` variable. These functions will extract components of *t*:

Function	Returns	Result if t = tc(05jul1972 21:38:02) (i.e., t = 394,839,482,000)
`hh`(*t*)	time of day, hours	21
`mm`(*t*)	time of day, minutes	38
`ss`(*t*)	time of day, seconds	2.000

If *t* is a `%tC` variable (corrected for leap seconds), then the corresponding functions to use are: `hhC`(*t*), `mmC`(*t*), and `ssC`(*t*).

Calculating Durations from Date and Time Variables

As mentioned above, all %t variables are really stored as durations from 1960:

Format	Units
`%tc`	milliseconds
`%tC`	milliseconds
`%td`	days
`%tw`	weeks
`%tm`	months
`%tq`	quarters
`%th`	half years

Thus, you can obtain the duration between two %t variables by subtracting them:

```
gen days_employed = currdate – hiredate
gen qtrs_to_jan15 = currqtr – qofd(td(15jan2011))
```

To add a duration to a date, add the two values:

```
gen lastdate = hiredate + days_employed
format lastdate %td

gen qtr_of_merger = currqtr + quarters_to_merger
format qtr_of_merger %tq
```

Formatting new date and time variables appropriately makes them readable in case you ever print them out or examine them in the data browser.

Also remember that durations in milliseconds, generated by adding or subtracting `%tc` and `%tC` variables, need to be stored as `doubles`.

```
gen double millisecs_employed = lasttime - hiretime
```

Stata also provides functionality to convert between time units:

Function	Purpose
`hours(`*ms*`)`	convert milliseconds to hours returns *ms*/(60 x 60 x 1000)
`minutes(`*ms*`)`	convert milliseconds to minutes returns *ms*/(60 x 1000)
`seconds(`*ms*`)`	convert milliseconds to seconds returns *ms*/1000
`msofhours(`*ms*`)`	convert hours to milliseconds returns *h* x 60 x 60 x 1000
`msofminutes(`*ms*`)`	convert minutes to milliseconds returns *m* x 60 x 1000
`msofseconds(`*ms*`)`	convert seconds to milliseconds returns *s* x 1000

Using these functions, you can code:

```
gen double days_employed = hours(lasttme – hiretime)/24

gen double lasttime = hiretime + msofh(24*days_employed)
```

Note that `days_employed` will include a fraction of a day. To round this to a whole number, use:

```
gen approx_days_employed = round(24*hours(lasttime-
hiretime))
```

For More Information and Assistance

STATA Help

Stata provides built-in Help pages, available by typing "`help date`" in the command box.

The *Stata 11 Data Management Guide*

This detailed reference is available for use in the Velma Denning Room in Green Library. The book does not circulate, but the room is open for reading weekdays 9 am – 5 pm.

Social Science Software Consulting

Software consultants are available during drop-in hours 3 pm – 5 pm Monday through Thursday throughout the academic quarter, and by appointment throughout the year. Please visit our website for more information or to make an appointment:

http://ssds.stanford.edu/

Note: This document is based on Stata 11 for Windows

Social Science Data and Software
Document revised: 7/6/10