Title

datetime — Date and time values and variables

Description

Syntax below provides a complete overview of Stata's date and time values. Also see [D] **datetime translation** and [D] **datetime display formats** for additional information.

Syntax

Syntax is presented under the following headings:

Also see

[D] **datetime translation**	String to numeric date translation functions
[D] **datetime display formats**	Display formats for dates and times

Types of dates and their human readable forms (HRFs)

Date type	Examples of HRFs
datetime	20jan2010 09:15:22.120
date	20jan2010, 20/01/2010, . . .
weekly date	2010w3
monthly date	2010m1
quarterly date	2010q1
half-yearly date	2010h1
yearly date	2010

The styles of the HRFs in the table above are merely examples. Perhaps you prefer 2010.01.20; Jan. 20, 2010; 2010-1; etc.

With the exception of yearly dates, HRFs are usually stored in string variables. If you are reading raw data, read the HRFs into strings.

HRFs are not especially useful except for reading by humans, and thus Stata provides another way of recording dates called Stata internal form (SIF). You can convert HRF dates to SIF.

Stata internal form (SIF)

The numeric values in the table below are equivalent to the string values in the table in the previous section.

SIF type	Examples in SIF	Units
datetime/c	1,579,598,122,120	milliseconds since 01jan1960 00:00:00.000, assuming 86,400 s/day
datetime/C	1,579,598,146,120	milliseconds since 01jan1960 00:00:00.000, adjusted for leap seconds*
date	18,282	days since 01jan1960 (01jan1960 = 0)
weekly date	2,601	weeks since 1960w1
monthly date	600	months since 1960m1
quarterly date	200	quarters since 1960q1
half-yearly date	100	half-years since 1960h1
yearly date	2010	years since 0000

* SIF datetime/C is equivalent to coordinated universal time (UTC). In UTC, leap seconds are periodically inserted because the length of the mean solar day is slowly increasing. See *Why there are two SIF datetime encodings* in [D] **datetime translation**.

SIF values are stored as regular Stata numeric variables.

You can convert HRFs into SIFs by using HRF-to-SIF conversion functions; see the next section, called *HRF-to-SIF conversion functions*.

You can make the numeric SIF readable by placing the appropriate *%fmt* on the numeric variable; see *Displaying SIFs in HRF*, below.

You can convert from one SIF type to another by using SIF-to-SIF conversion functions; see *SIF-to-SIF conversion*, below.

SIF dates are convenient because you can subtract them to obtain time between dates, for example,

$$\begin{aligned}
\text{datetime2} - \text{datetime1} &= \text{milliseconds between datetime1 and datetime2} \\
&\quad \text{(divide by 1,000 to obtain seconds)} \\
\text{date2} - \text{date1} &= \text{days between date1 and date2} \\
\text{week2} - \text{week1} &= \text{weeks between week1 and week2} \\
\text{month2} - \text{month1} &= \text{months between month1 and month2} \\
\text{half2} - \text{half1} &= \text{half-years between half1 and half2} \\
\text{year2} - \text{year1} &= \text{years between year1 and year2}
\end{aligned}$$

In the remaining text, we will use the following notation:

tc:	a Stata double variable containing SIF datetime/c values
tC:	a Stata double variable containing SIF datetime/C values
td:	a Stata variable containing SIF date values
tw:	a Stata variable containing SIF weekly date values
tm:	a Stata variable containing SIF monthly date values
tq:	a Stata variable containing SIF quarterly date values
th:	a Stata variable containing SIF half-yearly date values
ty:	a Stata variable containing SIF yearly date values

HRF-to-SIF conversion functions

SIF type	Function to convert HRF to SIF	Note
datetime/c	*tc* = `clock(`*HRFstr*`,` *mask*`)`	*tc* must be `double`
datetime/C	*tC* = `Clock(`*HRFstr*`,` *mask*`)`	*tC* must be `double`
date	*td* = `date(`*HRFstr*`,` *mask*`)`	*td* may be `float` or `long`
weekly date	*tw* = `weekly(`*HRFstr*`,` *mask*`)`	*tw* may be `float` or `int`
monthly date	*tm* = `monthly(`*HRFstr*`,` *mask*`)`	*tm* may be `float` or `int`
quarterly date	*tq* = `quarterly(`*HRFstr*`,` *mask*`)`	*tq* may be `float` or `int`
half-yearly date	*th* = `halfyearly(`*HRFstr*`,` *mask*`)`	*th* may be `float` or `int`
yearly date	*ty* = `yearly(`*HRFstr*`,` *mask*`)`	*ty* may be `float` or `int`

Warning: To prevent loss of precision, datetime SIFs must be stored as `double`s.

Examples:

1. You have datetimes stored in the string variable `mystr`, an example being "`2010.07.12 14:32`". To convert to SIF datetime/c, you type

```
. generate double eventtime = clock(mystr, "YMDhm")
```

   The mask `"YMDhm"` specifies the order of the datetime components. In this case, they are year, month, day, hour, and minute.

2. You have datetimes stored in `mystr`, an example being "`2010.07.12 14:32:12`". You type

```
. generate double eventtime = clock(mystr, "YMDhms")
```

   Mask element `s` specifies seconds. In example 1, there were no seconds; in this example, there are.

3. You have datetimes stored in `mystr`, an example being "`2010 Jul 12 14:32`". You type

```
. generate double eventtime = clock(mystr, "YMDhm")
```

   This is the same command that you typed in example 1. In the mask, you specify the order of the components; Stata figures out the style for itself. In example 1, months were numeric. In this example, they are spelled out (and happen to be abbreviated).

4. You have datetimes stored in `mystr`, an example being "`July 12, 2010 2:32 PM`". You type

   ```
   . generate double eventtime = clock(mystr, "MDYhm")
   ```

   Stata automatically looks for AM and PM, in uppercase and lowercase, with and without periods.

5. You have datetimes stored in `mystr`, an example being "`7-12-10 14.32`". The 2-digit year is to be interpreted as being prefixed with 20. You type

   ```
   . generate double eventtime = clock(mystr, "MD20Yhm")
   ```

6. You have datetimes stored in `mystr`, an example being "`14:32 on 7/12/2010`". You type

   ```
   . generate double eventtime = clock(mystr, "hm#MDY")
   ```

   The `#` sign between `m` and `M` means, "ignore one thing between minute and month", which in this case is the word "on". Had you omitted the `#` from the mask, the new variable `eventtime` would have contained missing values.

7. You have a date stored in `mystr`, an example being "`22/7/2010`". In this case, you want to create an SIF date instead of a datetime. You type

   ```
   . generate eventdate = date(mystr, "DMY")
   ```

   Typing

   ```
   . generate double eventtime = clock(mystr, "DMY")
   ```

   would have worked, too. Variable `eventtime` would contain a different coding from that contained by `eventdate`; namely, it would contain milliseconds from 1jan1960 rather than days (1,595,376,000,000 rather than 18,465). Datetime value 1,595,376,000,000 corresponds to 22jul2010 00:00:00.000.

See [D] **datetime translation** for more information about the HRF-to-SIF conversion functions.

Displaying SIFs in HRF

SIF type	Display format to present SIF in HRF
datetime/c	`%tc`
datetime/C	`%tC`
date	`%td`
weekly date	`%tw`
monthly date	`%tm`
quarterly date	`%tq`
half-yearly date	`%th`
yearly date	`%ty`

The display formats above are the simplest forms of each of the SIFs. You can control how each type of SIF date is displayed; see [D] **datetime display formats**.

Examples:

1. You have datetimes stored in string variable `mystr`, an example being "`2010.07.12 14:32`". To convert to SIF datetime/c and make the new variable readable when displayed, you type

```
. generate double eventtime = clock(mystr, "YMDhm")
. format eventtime %tc
```

2. You have a date stored in `mystr`, an example being "`22/7/2010`". To convert to an SIF date and make the new variable readable when displayed, you type

```
. generate eventdate = date(mystr, "DMY")
. format eventdate %td
```

Building SIFs from components

SIF type	Function to build from components
datetime/c	*tc* = `mdyhms(`*M*`,` *D*`,` *Y*`,` *h*`,` *m*`,` *s*`)`
	tc = `dhms(`*td*`,` *h*`,` *m*`,` *s*`)`
	tc = `hms(`*h*`,` *m*`,` *s*`)`
datetime/C	*tC* = `Cmdyhms(`*M*`,` *D*`,` *Y*`,` *h*`,` *m*`,` *s*`)`
	tC = `Cdhms(`*td*`,` *h*`,` *m*`,` *s*`)`
	tC = `Chms(`*h*`,` *m*`,` *s*`)`
date	*td* = `mdy(`*M*`,` *D*`,` *Y*`)`
weekly date	*tw* = `yw(`*Y*`,` *W*`)`
monthly date	*tm* = `ym(`*Y*`,` *M*`)`
quarterly date	*tq* = `yq(`*Y*`,` *Q*`)`
half-yearly date	*th* = `yh(`*Y*`,` *H*`)`
yearly date	*ty* = `y(`*Y*`)`

Warning: SIFs for datetimes must be stored as `double`s.

Examples:

1. Your dataset has three variables, `mo`, `da`, and `yr`, with each variable containing a date component in numeric form. To convert to SIF date, you type

```
. generate eventdate = mdy(mo, da, yr)
. format eventdate %td
```

2. Your dataset has two numeric variables, `mo` and `yr`. To convert to SIF date corresponding to the first day of the month, you type

```
. generate eventdate = mdy(mo, 1, yr)
. format eventdate %td
```

3. Your dataset has two numeric variables, `da` and `yr`, and one string variable, `month`, containing the spelled-out month. In this case, do not use the building-from-component functions. Instead, construct a new string variable containing the HRF and then convert the string using the HRF-to-SIF conversion functions:

```
. generate str work  = month + " " + string(da) + " " + string(yr)
. generate eventdate = date(work, "MDY")
. format eventdate %td
```

SIF-to-SIF conversion

From:	To: datetime/c	datetime/C	date
datetime/c		tC = `Cofc(`*tc*`)`	td = `dofc(`*tc*`)`
datetime/C	tc = `cofC(`*tC*`)`		td = `dofC(`*tC*`)`
date	tc = `cofd(`*td*`)`	tC = `Cofd(`*td*`)`	
weekly			td = `dofw(`*tw*`)`
monthly			td = `dofm(`*tm*`)`
quarterly			td = `dofq(`*tq*`)`
half-yearly			td = `dofh(`*th*`)`
yearly			td = `dofy(`*ty*`)`

From:	To: weekly	monthly	quarterly
date	tw = `wofd(`*td*`)`	tm = `mofd(`*td*`)`	tq = `qofd(`*td*`)`

From:	To: half-yearly	yearly
date	th = `hofd(`*td*`)`	ty = `yofd(`*td*`)`

To convert between missing entries, use two functions, going through date or datetime as appropriate. For example, quarterly of monthly is `tq` = `qofd(dofm(`*tm*`))`.

Examples:

1. You have the SIF datetime/c variable **eventtime** and wish to create the new variable **eventdate** containing just the date from the datetime variable. You type

```
. generate eventdate = dofc(eventtime)
. format eventdate %td
```

2. You have the SIF date variable **eventdate** and wish to create the new SIF datetime/c variable **eventtime** from it. You type

```
. generate double eventtime = cofd(eventdate)
. format eventtime %tc
```

   The time components of the new variable will be set to the default 00:00:00.000.

3. You have the SIF quarterly variable **eventqtr** and wish to create the new SIF date variable **eventdate** from it. You type

```
. generate eventdate = dofq(eventqtr)
. format eventdate %tq
```

   The new variable, **eventdate**, will contain 01jan dates for quarter 1, 01apr dates for quarter 2, 01jul dates for quarter 3, and 01oct dates for quarter 4.

4. You have the SIF datetime/c variable **admittime** and wish to create the new SIF quarterly variable **admitqtr** from it. You type

```
. generate admitqtr = qofd(dofc(admittime))
. format admitqtr %tq
```

   Because there is no `qofc()` function, you use `qofd(dofc())`.

Extracting time-of-day components from SIFs

Desired component	Function	Example
hour of day	`hh(`*tc*`)` or `hhC(`*tC*`)`	14
minutes of day	`mm(`*tc*`)` or `mmC(`*tC*`)`	42
seconds of day	`ss(`*tc*`)` or `ssC(`*tC*`)`	57.123

Notes:

$0 \le$ `hh(`*tc*`)` ≤ 23, $0 \le$ `hhC(`*tC*`)` ≤ 23
$0 \le$ `mm(`*tc*`)` ≤ 59, $0 \le$ `mmC(`*tC*`)` ≤ 59
$0 \le$ `ss(`*tc*`)` < 60, $0 \le$ `ssC(`*tC*`)` < 61 (sic)

Example:

1. You have the SIF datetime/c variable `admittime`. You wish to create the new variable `admithour` equal to the hour and fraction of hour within the day of admission. You type

```
        . generate admithour = hh(admittime) + mm(admittime)/60
        > + ss(admittime)/3600
```

Extracting date components from SIFs

Desired component	Function	Example*
calendar year	`year(`*td*`)`	2013
calendar month	`month(`*td*`)`	7
calendar day	`day(`*td*`)`	5
day of week (0=Sunday)	`dow(`*td*`)`	2
Julian day of year (1=first day)	`doy(`*td*`)`	186
week within year (1=first week)	`week(`*td*`)`	27
quarter within year (1=first quarter)	`quarter(`*td*`)`	3
half within year (1=first half)	`halfyear(`*td*`)`	2

* All examples are with `td=mdy(7,5,2013)`.
All functions require an SIF date as an argument. To extract components from other SIFs, use the appropriate SIF-to-SIF conversion function to convert to an SIF date, for example, `quarter(dofq(tq))`.

Examples:

1. You wish to obtain the day of week Sunday, Monday, ..., corresponding to the SIF date variable `eventdate`. You type

```
        . generate day_of_week = dow(eventdate)
```

The new variable, `day_of_week`, contains 0 for Sunday, 1 for Monday, ..., 6 for Saturday.

2. You wish to obtain the day of week Sunday, Monday, . . . , corresponding to the SIF datetime/c variable `eventtime`. You type

   ```
   . generate day_of_week = dow(dofc(eventtime))
   ```

3. You have the SIF date variable `evdate` and wish to create the new SIF date variable `evdate_r` from it. `evdate_r` will contain the same date as `evdate` but rounded back to the first of the month. You type

   ```
   . generate evdate_r = mdy(month(evdate), 1, year(evdate))
   ```

   In the above solution, we used the date-component extraction functions `month()` and `year()` and used the build-from-components function `mdy()`.

Conveniently typing SIF values

You can type SIF values by just typing the number, such as 16,237 or 1,402,920,000,000, as in

```
. generate before = cond(hiredon < 16237, 1, 0) if !missing(hiredon)
. drop if admittedon < 1402920000000
```

Easier to type is

```
. generate before = cond(hiredon < td(15jun2004), 1, 0) if !missing(hiredon)
. drop if admittedon < tc(15jun2004 12:00:00)
```

You can type SIF date values by typing the date inside `td()`, as in `td(15jun2004)`.

You can type SIF datetime/c values by typing the datetime inside `tc()`, as in `tc(15jun2004 12:00:00)`.

`td()` and `tc()` are called pseudofunctions because they translate what you type into their numerical equivalents. Pseudofunctions require only that you specify the datetime components in the expected order, so rather than 15jun2004 above, we could have specified 15 June 2004, 15-6-2004, or 15/6/2004.

The SIF pseudofunctions and their expected component order are

Desired SIF type	Pseudofunction
datetime/c	`tc(` [*day-month-year*] *hh*:*mm*[:*ss* [.*sss*]]`)`
datetime/C	`tC(` [*day-month-year*] *hh*:*mm*[:*ss* [.*sss*]]`)`
date	`td(`*day-month-year*`)`
weekly date	`tw(`*year-week*`)`
monthly date	`tm(`*year-month*`)`
quarterly date	`tq(`*year-quarter*`)`
half-yearly date	`th(`*year-half*`)`
yearly date	none necessary; just type year

The *day-month-year* in `tc()` and `tC()` are optional. If you omit them, 01jan1960 is assumed. Doing so produces time as an offset, which can be useful in, for example,

```
. generate six_hrs_later = eventtime + tc(6:00)
```

Obtaining and working with durations

SIF values are simply durations from 1960. SIF datetime/c values record the number of milliseconds from 1jan1960 00:00:00; SIF date values record the number of days from 1jan1960, and so on.

To obtain the time between two SIF variables—the duration—subtract them:

```
. generate days_employed = curdate - hiredate
. generate double ms_inside = discharge_time - admit_time
```

To obtain a new SIF that is equal to an old SIF before or after some amount of time, just add or subtract the desired durations:

```
. generate lastdate = hiredate + days_employed
. format lastdate %td
. generate double admit_time = discharge_time - ms_inside
. format admit_time %tc
```

Remember to use the units of the SIF variables. SIF dates are in terms of days, SIF weekly dates are in terms of weeks, etc., and SIF datetimes are in terms of milliseconds. Concerning milliseconds, it is often easier to use different units and conversion functions to convert to milliseconds:

```
. generate hours_inside = hours(discharge_time - admit_time)
. generate admit_time = discharge_time - msofhours(hours_inside)
. format admit_time %tc
```

Function `hours()` converts milliseconds to hours. Function `msofhours()` converts hours to milliseconds. The millisecond conversion functions are

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

If you plan on using returned values to add to or subtract from a datetime SIF, be sure they are stored as `double`s.

Using dates and times from other software

Most software stores dates and times numerically as durations from some sentinel date in specified units, but they differ on the sentinel date and the units. If you have imported data, it is usually possible to adjust the numeric date and datetime values to SIF.

Converting SAS dates:

SAS provides dates measured as the number of days since 01jan1960. This is the same coding as used by Stata:

```
. generate statadate = sasdate
. format statadate %td
```

SAS provides datetimes measured as the number of seconds since 01jan1960 00:00:00, assuming 86,400 seconds/day. To convert to SIF datetime/c, type

```
. generate double statatime = (sastime*1000)
. format statatime %tc
```

It is important that variables containing SAS datetimes, such as `sastime` above, be imported into Stata as `double`s.

Converting SPSS dates:

SPSS provides dates and datetimes measured as the number of seconds since 14oct1582 00:00:00, assuming 86,400 seconds/day. To convert to SIF datetime/c, type

```
. generate double statatime = (spsstime*1000) + tc(14oct1582 00:00)
. format statatime %tc
```

To convert to SIF date, type

```
. generate statadate = dofc((spsstime*1000) + tc(14oct1582 00:00))
. format statadate %td
```

Converting R dates:

R stores dates as days since 01jan1970. To convert to SIF date, type

```
. generate statadate = rdate - td(01jan1970)
. format statadate %td
```

R stores datetimes as the number of UTC-adjusted seconds since 01jan1970 00:00:00. To convert to SIF datetime/C, type

```
. generate double statatime = rtime - tC(01jan1970 00:00)
. format statatime %tC
```

To convert to SIF datetime/c, type

```
. generate double statatime = cofC(rtime - tC(01jan1970 00:00))
. format statatime %tc
```

There are issues of which you need to be aware when working with datetime/C values; see *Why there are two SIF datetime encodings* and *Advice on using datetime/c and datetime/C*, both in [D] **datetime translation**.

Converting Excel dates:

If you have data in an Excel format file, you may want to use the `import excel` command. If the Excel file contains numerically encoded dates, `import excel` will read those dates and properly code them in SIF. You do not need to perform any conversion after importing your data with `import excel`.

On the other hand, if you copy and paste a spreadsheet into Stata's editor, dates and datetimes are pasted as strings in HRF. The discussion below concerns converting such HRF datetime strings to SIF numeric values.

Excel has used different date systems across operating systems. Excel for Windows used the "1900 Date System". Excel for Mac used the "1904 Date System". More recently, Excel has been standardizing on the 1900 Date System on all operating systems.

Regardless of operating system, Excel can use either encoding. See http://support.microsoft.com/kb/214330 for instructions on converting workbooks between date systems.

Converted dates will be off by four years if you choose the wrong date system.

Converting Excel 1900-Date-System dates:

For dates on or after 01mar1900, Excel stores dates as days since 30dec1899. To convert to a Stata date,

```
. generate statadate = exceldate + td(30dec1899)
. format statadate %td
```

Excel can store dates between 01jan1900 and 28feb1900, but the formula above will not handle those two months. See http://www.cpearson.com/excel/datetime.htm for more information.

For datetimes on or after 01mar1900 00:00:00, Excel stores datetimes as days plus fraction of day since 30dec1899 00:00:00. To convert with a one-second resolution to a Stata datetime,

```
. generate statatime = round((exceltime+td(30dec1899))*86400)*1000
. format statatime %tc
```

Converting Excel 1904-Date-System dates:

For dates on or after 01jan1904, Excel stores dates as days since 01jan1904. To convert to a Stata date,

```
. generate statadate = exceldate + td(01jan1904)
. format statadate %td
```

For datetimes on or after 01jan1904 00:00:00, Excel stores datetimes as days plus fraction of day since 01jan1904 00:00:00. To convert with a one-second resolution to a Stata datetime,

```
. generate statatime = round((exceltime+td(01jan1904))*86400)*1000
. format statatime %tc
```

Converting OpenOffice dates:

OpenOffice uses the Excel 1900 Date System described above.

Converting Unix time:

Unix time is stored as the number of seconds since midnight, 01jan1970. To convert to a Stata datetime,

```
    . generate double statatime = unixtime * 1000 + mdyhms(1,1,1970,0,0,0)
```

To convert to a Stata date,

```
    . generate statadate = dofc(unixtime * 1000 + mdyhms(1,1,1970,0,0,0))
```

Remarks and examples stata.com

The best way to learn about Stata's date and time functions is to experiment with them using the `display` command; see [P] **display**.

```
. display date("5-12-1998", "MDY")
14011

. display %td date("5-12-1998", "MDY")
12may1998

. display clock("5-12-1998 11:15", "MDY hm")
1.211e+12

. display %20.0gc clock("5-12-1998 11:15", "MDY hm")
1,210,590,900,000

. display %tc clock("5-12-1998 11:15", "MDY hm")
12may1998 11:15:00
```

With `display`, you can specify a format in front of the expression to specify how the result is to be formatted.

References

Cox, N. J. 2010. Stata tip 68: Week assumptions. *Stata Journal* 10: 682–685.

——. 2012. Stata tip 111: More on working with weeks. *Stata Journal* 12: 565–569.

Dyck, A. 2011. Working with Unix timestamps in Stata. Statabytes. http://statabytes.andrewdyck.com/blog/working-with-unix-timestamps-in-stata/.

Gould, W. W. 2011. Using dates and times from other software. The Stata Blog: Not Elsewhere Classified. http://blog.stata.com/2011/01/05/using-dates-and-times-from-other-software/.

Also see

[D] **datetime business calendars** — Business calendars

[D] **datetime display formats** — Display formats for dates and times

[D] **datetime translation** — String to numeric date translation functions