Datetime — Date and time values and variables

Description Quick start Syntax Remarks and examples References Also see

Description

This entry provides a complete overview of Stata's date and time values. We discuss functions used to obtain Stata dates, including string-to-numeric conversions and conversions among different types of dates and times.

Stata's date and time values need to be formatted so they look like the dates and times we are familiar with. We show basic formatting options here, but more details can be found in [D] **Datetime display formats**.

[D] **Datetime conversion** has more details on converting dates and times stored as strings to numerically encoded Stata dates and times.

[D] **Datetime values from other software** discusses getting Stata dates from dates created by other software.

[D] **Datetime durations** describes functions designed to get durations (for example, ages) from two Stata dates or to express a duration in different units.

[D] **Datetime relative dates** describes functions that return dates based on other dates, for example, the date of a birthday in another year.

[D] **Datetime business calendars** describes business calendars—using dates with nonbusiness days (for example, weekends and holidays) removed. You can use existing calendars or create your own; see [D] **Datetime business calendars creation**.

For an alphabetical listing of all the datetime functions, see [FN] **Date and time functions**.

Quick start

Convert the string variable `strdate`, with dates such as `"January 1, 2020"`, to a numerically encoded Stata date

```
generate numdate = date(strdate, "MDY")
```

Format `numdate` to make it readable when displayed

```
format numdate %td
```

Convert the string variable `strtime`, with dates and times such as `"January 1,2020 10:30 am"`, to a numerically encoded Stata datetime variable

```
generate double numtime = clock(strtime, "MDYhm")
```

Format `numtime` to make it readable when displayed

```
format numtime %tc
```

Convert the string variable strmonthly, with monthly dates such as "2012-04", to a Stata date, and format it to make it readable when displayed

```
generate nummonth = monthly(strmonthly, "YM")
format nummonth %tm
```

List observations for which numdate is prior to February 15, 2013

```
list if numdate < td(15/2/2013)
```

Create a monthly date variable from numeric variables year and month

```
generate monthly = ym(year,month)
```

Create a daily date variable from the datetimes stored in numtime

```
generate dateoftime = dofc(numtime)
```

Create a monthly date variable from the daily dates stored in numdate

```
generate monthlyofdate = mofd(numdate)
```

Create a new variable with the month of the daily dates stored in numdate

```
generate monthnum = month(numdate)
```

Syntax

Syntax is presented under the following headings:

Types of dates and how they are displayed
How Stata dates are stored
Converting dates stored as strings to Stata dates
Formatting Stata dates for display
Creating dates from components
Converting among units
Extracting time-of-day components from datetimes
Extracting date components from daily dates
Typing dates into expressions

Types of dates and how they are displayed

Dates and times can take many forms; below, we list the types of dates that are supported in Stata. Note that throughout our documentation, we use the term "datetime" to refer to variables that record time or date and time.

Date type	Examples
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 dates in the table above are merely examples; dates can be displayed in a number of ways. Perhaps you prefer 2010.01.20; Jan. 20, 2010; 2010-1; etc.

How Stata dates are stored

Stata dates are numeric values that record durations (positive or negative) from 01jan1960. Below, we list the numeric values corresponding to the dates displayed in the table in the previous section.

Stata date type	Examples	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

* 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 datetime encodings* in [D] **Datetime conversion**.

Stata dates are stored as regular Stata numeric variables.

You can convert dates stored as strings to Stata dates by using the string-to-numeric conversion functions; see *Converting dates stored as strings to Stata dates*.

You can make Stata dates readable by placing the appropriate *%fmt* on the numeric variable; see *Formatting Stata dates for display*.

You can convert from one Stata date type to another by using conversion functions; see *Converting among units*.

Storing dates as numeric values is convenient because you can subtract them to obtain time between dates, for example,

datetime2 − datetime1	= milliseconds between datetime1 and datetime2 (divide by 1,000 to obtain seconds)
date2 − date1	= days between date1 and date2
week2 − week1	= weeks between week1 and week2
month2 − month1	= months between month1 and month2
half2 − half1	= half-years between half1 and half2
year2 − year1	= years between year1 and year2

For time differences in other units, for example, the number of years between date1 and date2, see [D] **Datetime durations**.

Converting dates stored as strings to Stata dates

To convert dates and times stored as strings to Stata dates and times, use one of the functions listed below.

Stata date type	Function	Required variable precision
datetime/c	`clock(`*str*`, `*mask*`)`	`double`
datetime/C	`Clock(`*str*`, `*mask*`)`	`double`
date	`date(`*str*`, `*mask*`)`	`float` or `long`
weekly date	`weekly(`*str*`, `*mask*`)`*	`float` or `int`
monthly date	`monthly(`*str*`, `*mask*`)`*	`float` or `int`
quarterly date	`quarterly(`*str*`, `*mask*`)`*	`float` or `int`
half-yearly date	`halfyearly(`*str*`, `*mask*`)`*	`float` or `int`
yearly date	`yearly(`*str*`, `*mask*`)`	`float` or `int`

* *str* is a string variable or a literal string enclosed in quotes.

Within each function, you need to specify the string you want to convert and the order in which the date and time components appear in that string.

The string to be converted with `clock()`, `Clock()`, and `date()` may contain dates and times that are run together or include punctuation marks between the components. However, the functions marked with an asterisk require that the string date contain a space or punctuation between the year and the other component if the string consists only of numbers. For more information on how punctuation is handled and other details related to these conversion functions, see [D] **Datetime conversion**.

The order of the components is specified within quotes, such as `"YMD"`, and is referred to as a mask. The mask may contain the following elements:

Mask element	Component
`D`	day
`W`	week
`M`	month
`Q`	quarter
`H`	half-year
`Y`	year
`19Y`	two-digit year in the 1900s
`20Y`	two-digit year in the 2000s
`h`	hour
`m`	minute
`s`	second
`#`	placeholder for something to be ignored

Examples:

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

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

   The string contains the year, month, and day followed by the hour and minute, so you specify the mask "YMDhm".

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 a Stata 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.

Formatting Stata dates for display

While Stata dates are stored as regular Stata numeric variables, they are formatted so they look like the dates and times we are familiar with. Each type of date has a corresponding display format, and we list them below:

Stata date type	Display format
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 Stata dates. You can control how each type of Stata 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 this to a Stata datetime/c variable 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 this to a Stata date variable and make the new variable readable when displayed, you type

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

Creating dates from components

If you have components of your date stored separately, you can use the following functions to create a single date variable. Note that each component used in this function must be numeric; you can specify numeric variables or simply digits.

Stata date type	Function to build from components
datetime/c	`mdyhms(`*M*`,` *D*`,` *Y*`,` *h*`,` *m*`,` *s*`)`*
	`dhms(`e_d`,` *h*`,` *m*`,` *s*`)`*†
	`hms(`*h*`,` *m*`,` *s*`)`*
datetime/C	`Cmdyhms(`*M*`,` *D*`,` *Y*`,` *h*`,` *m*`,` *s*`)`*
	`Cdhms(`e_d`,` *h*`,` *m*`,` *s*`)`*†
	`Chms(`*h*`,` *m*`,` *s*`)`*
date	`mdy(`*M*`,` *D*`,` *Y*`)`
	`dmy(`*D*`,` *M*`,` *Y*`)`
weekly date	`yw(`*Y*`,` *W*`)`
monthly date	`ym(`*Y*`,` *M*`)`
quarterly date	`yq(`*Y*`,` *Q*`)`
half-yearly date	`yh(`*Y*`,` *H*`)`
yearly date	`y(`*Y*`)`

* Stata datetime variables must be stored as `doubles`.
† e_d is a Stata date with a month, day, and year component.

Examples:

1. Your dataset has three variables, `mo`, `da`, and `yr`, with each variable containing a date component in numeric form. To create a date variable from these components, you type

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

   If you prefer the ordering day, month, and year, you can use `dmy()` instead of `mdy()`:

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

2. Your dataset has two numeric variables, `mo` and `yr`. To create a date variable 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 with these components, and then convert the string to a Stata date using the conversion functions:

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

Converting among units

The table below lists the functions for converting one type of date and time to another. Because there are not official functions for every possible conversion, we have also included the functions you can nest instead to obtain those conversions. Similarly, for any other conversion not listed here, you can use two functions, going through date or datetime as appropriate. For example, to obtain a monthly date from a datetime/c variable, you would use `mofd(dofc(`*varname*`))`.

	To:		
From:	datetime/c	datetime/C	date
datetime/c		`Cofc()`	`dofc()`
datetime/C	`cofC()`		`dofC()`
date	`cofd()`	`Cofd()`	

	To:			
From:	date	weekly	monthly	quarterly
date		`wofd()`	`mofd()`	`qofd()`
weekly	`dofw()`		`mofd(dofw())`	`qofd(dofw())`
monthly	`dofm()`	`wofd(dofm())`		`qofd(dofm())`
quarterly	`dofq()`	`wofd(dofq())`	`mofd(dofq())`	

	To:		
From:	date	half-yearly	yearly
date		`hofd()`	`yofd()`
half-yearly	`dofh()`		
yearly	`dofy()`		

Note that if you are converting to a date type for which you do not have all the components, those missing elements will be set to their defaults. For example, converting a yearly date to a weekly date would give you the first week of each year. Converting a quarterly date to a monthly date would give you the first month of each quarter, along with the year, of course. Below, we list the defaults for the date and time components:

Date component	Default
`year`	1960
`half-year`	1
`quarter`	1
`month`	1
`week`	1
`day`	01
`hour`	00
`minute`	00
`second`	00

Examples:

1. You have the Stata 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 daily date `eventdate` and wish to create the new datetime/c variable `eventtime` from it. For this unusual case, you can even 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 Stata quarterly variable `eventqtr` and wish to create the new Stata 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 datetime/c variable `admittime` and wish to create the 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 datetimes

In the table below, we list the functions used to extract time-of-day components from datetimes. If you are working with standard datetimes, use the functions in the datetime/c column. If you are working with leap second–adjusted times, use the functions in the datetime/C column.

	Function		
Desired component	datetime/c	datetime/C	Example
hour of day	`hh(`e_{tc}`)`	`hhC(`e_{tC}`)`	14
minutes of day	`mm(`e_{tc}`)`	`mmC(`e_{tC}`)`	42
seconds of day	`ss(`e_{tc}`)`	`ssC(`e_{tC}`)`	57.123
year, month, day, hour, minute, second, or millisecond	`clockpart(`e_{tc}`,`s_u`)`	`Clockpart(`e_{tC}`,`s_u`)`	2020

e_{tc} is a Stata datetime/c value.
e_{tC} is a Stata datetime/C value (UTC time with leap seconds).
s_u is a string specifying the time unit. s_u can be string `"year"` or `"y"` for year; `"month"` or `"mon"` for month; `"day"` or `"d"` for day; `"hour"` or `"h"` for hour; `"minute"` or `"min"` for minute; `"second"`, `"sec"`, or `"s"` for second; and `"millisecond"` or `"ms"` for millisecond (case insensitive).
Notes:

$0 \leq$ `hh(`e_{tc}`)` ≤ 23, $0 \leq$ `hhC(`e_{tC}`)` ≤ 23
$0 \leq$ `mm(`e_{tc}`)` ≤ 59, $0 \leq$ `mmC(`e_{tC}`)` ≤ 59
$0 \leq$ `ss(`e_{tc}`)` < 60, $0 \leq$ `ssC(`e_{tC}`)` < 61 (sic)

Example:

1. You have the Stata 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
```

2. You have the Stata datetime/C variable `admitTime`. You wish to create the new variable `admityear` to record the year of admission. You type

```
. generate admityear = Clockpart(admitTime, "year")
```

See [D] **Datetime durations** for other functions that can be used to calculate durations.

Extracting date components from daily dates

You might be working with dates that have more information than you need. For example, daily dates refer to dates that have a month, day, and year component. If you want to refer only to the month, or year, of a daily date, you can use the extraction functions below.

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

* e_d is a Stata date with a month, day, and year component.
† All examples are with e_d = `mdy(7,5,2013)`.

All functions require a numeric Stata daily date as an argument. A string variable cannot be specified as the date. To extract components from other Stata date types, use the appropriate conversion function to convert to a daily date. For example, `quarter(dofq(`*qvar*`))` would return the quarter of the quarterly date values stored in *qvar*.

Examples:

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

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

3. You have the daily date variable `evdate` and wish to create the new 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()`.

Typing dates into expressions

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

You can type Stata 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 pseudofunctions and their expected component order are

Desired date 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; years are numeric and can be typed directly

Note that 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)
```

Note that string-to-date functions can be used in expressions with literal strings. For example, `date("15jun2004","DMY")` gives the same result as `td(15jun2004)`.

Remarks and examples

Remarks are presented under the following headings:

Introduction
Example 1: Converting string datetimes to Stata datetimes
Example 2: Extracting date components
Example 3: Building dates from components
Example 4: Converting among date types
Example 5: Using dates in expressions

Introduction

To use dates in Stata, you must first convert what you have to a Stata date. Stata dates are numbers, so they can easily be translated from, say, daily dates to monthly dates. Even so, they can be formatted so that they look like the dates you are familiar with. If you have dates stored as strings, you must first convert them to Stata dates.

Converting a string date to a Stata date is as simple as telling Stata the string date and the order of the components. For example, we have a fictional dataset on patients who visited a local hospital. We have their birthdates, the dates of their visits, the reasons for their visits, and the dates they were discharged. All dates and times are stored as strings.

```
. use https://www.stata-press.com/data/r19/visits
(Fictional hospital visit data)
. describe
Contains data from https://www.stata-press.com/data/r19/visits.dta
 Observations:             5                  Fictional hospital visit data
    Variables:             7                  27 Aug 2024 22:56
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
patid           byte    %9.0g                 Patient ID
dateofbirth     str9    %9s                   Date of birth
reason          str15   %15s                  Reason for visit
admit_d         str8    %9s                   Admission date
admit_t         str17   %17s                  Admission date and time
discharge_d     str9    %9s                   Discharge date
discharge_t     str14   %14s                  Discharge date and time
-------------------------------------------------------------------------------
Sorted by:
```

```
. list admit_d dateofbirth
```

	admit_d	dateofb~h
1.	20110625	May152001
2.	20110313	Apr011999
3.	20110409	Nov151975
4.	20120211	Aug261960
5.	20120801	Dec161987

If we wanted to sort our data by birthdates or use these dates to compute a patient's age, we would need these variables to be numeric, not strings. So let's create numeric Stata dates from the birthdates and dates of admission:

```
. generate admit = date(admit_d, "YMD")
. generate dob = date(dateofbirth, "MDY")
. list admit_d admit dateofbirth dob
```

	admit_d	admit	dateofb~h	dob
1.	20110625	18803	May152001	15110
2.	20110313	18699	Apr011999	14335
3.	20110409	18726	Nov151975	5797
4.	20120211	19034	Aug261960	238
5.	20120801	19206	Dec161987	10211

For dates of admission, we told Stata that the string date was stored in `admit_d` and that the date was stored in the following order: year, month, day (YMD). Similarly, for birthdates we specify the string date and the order of the components: month, day, and year (MDY). It does not matter whether the month is written as a number, spelled out completely, or abbreviated to three letters.

You might be surprised by the values listed. The numbers represent the days elapsed since January 1, 1960, Stata's base date. Most software store dates and times in this manner, but they differ in the date they choose as a base. For us to understand the dates that these values represent, we apply a display format. All datetime display formats begin with a `%t` and contain a second letter representing the type of date: `%td` for daily dates, `%tw` for weekly dates, and so on. In our case, we have daily dates, so we use the `%td` format.

```
. format admit dob %td
. list admit dob
```

	admit	dob
1.	25jun2011	15may2001
2.	13mar2011	01apr1999
3.	09apr2011	15nov1975
4.	11feb2012	26aug1960
5.	01aug2012	16dec1987

If we instead had weekly dates, monthly dates, or quarterly dates, we would use the appropriate string-to-numeric conversion function to create the numeric variable and the appropriate display format. For more ways to format the dates above, see [D] **Datetime display formats**.

This is a simple example to get us started. The key points are that we want our dates to be stored numerically and formatted so that they look like the dates we are familiar with.

Below, we will discuss how to work with other types of dates. We will explore dates that have a time component, dates with components stored in multiple variables, and dates that have more components than we wish to work with. So whether you need to build, extract, or convert among different types of dates, you will learn how to do so with the examples that follow.

Example 1: Converting string datetimes to Stata datetimes

In this dataset, we also have string variables that record the date and time of admission and discharge:

```
. codebook admit_t discharge_t
```

```
-------------------------------------------------------------------------------
admit_t                                                 Admission date and time
-------------------------------------------------------------------------------

                  Type: String (str17)
         Unique values: 5                         Missing "": 0/5
            Tabulation: Freq.  Value
                        1  "20110313 8:30:45"
                        1  "20110409 10:17:08"
                        1  "20110625 5:15:06"
                        1  "20120211 10:30:12"
                        1  "20120801 6:45:59"
               Warning: Variable has embedded blanks.

-------------------------------------------------------------------------------
discharge_t                                         Discharge date and time
-------------------------------------------------------------------------------

                  Type: String (str14)
         Unique values: 5                         Missing "": 0/5
            Tabulation: Freq.  Value
                        1  "20110326 2:15"
                        1  "20110409 19:35"
                        1  "20110629 10:27"
                        1  "20120216 2:15"
                        1  "20120802 11:59"
               Warning: Variable has embedded blanks.
```

Let's convert these to Stata dates. Regardless if we are working with simple dates or dates and times, the process is the same. We are going to specify the string we want to convert and the order of the components. The only difference between this example and the previous example is the function; because these variables record the date and time, we will now use the `clock()` function, and the variables we generate will be referred to as datetime variables.

```
. generate double admit_time = clock(admit_t, "YMDhms")
. generate double disch_time = clock(discharge_t, "YMDhm")
. format admit_time disch_time %tc
. list admit_time disch_time
```

	admit_time	disch_time
1.	25jun2011 05:15:06	29jun2011 10:27:00
2.	13mar2011 08:30:45	26mar2011 02:15:00
3.	09apr2011 10:17:08	09apr2011 19:35:00
4.	11feb2012 10:30:12	16feb2012 02:15:00
5.	01aug2012 06:45:59	02aug2012 11:59:00

Note that the string variable `admit_t` contained the hour, minutes, and seconds, whereas the string variable `discharge_t` contained only the hour and minutes. This is why we did not specify an `s` in the list of components for `discharge_t`, and it is also why the seconds are set to zero for `disch_time`.

These variables now record the milliseconds since 01jan1960 00:00:00.000, assuming 86,400 seconds per day. You might have guessed that these values will be quite large, which is why we need to use the most precise storage type in Stata, `double`.

We have a lot of information in these variables, but we can choose to view just the portion in which we are interested by modifying the display format. For example, below we specify that we want to display only the hour and minute for the time of discharge, and we list the newly formatted time alongside the original string variable.

```
. format disch_time %tcHH:MM
. list discharge_t disch_time
```

	discharge_t	disch_~e
1.	20110629 10:27	10:27
2.	20110326 2:15	02:15
3.	20110409 19:35	19:35
4.	20120216 2:15	02:15
5.	20120802 11:59	11:59

We created the datetime variables above assuming there are 86,400 seconds in a day. This is one way to record time; another way would be to use UTC. UTC times are adjusted for leap seconds and can be obtained by modifying our commands just slightly, as follows:

```
. generate double admit_Time = Clock(admit_t, "YMDhms")
. format admit_Time %tC
```

Notice that the `Clock()` function and the `%tC` display format both contain a capital C. When you are working with standard datetimes, you will use functions with a lowercase `c`, and for UTC times, you will use functions with an uppercase `C`.

Example 2: Extracting date components

Suppose we want to work with just the month or year of admission. We can extract these components from our Stata date variable:

```
. generate admonth = month(admit)
. generate adyear = year(admit)
. list admit admonth adyear
```

	admit	admonth	adyear
1.	25jun2011	6	2011
2.	13mar2011	3	2011
3.	09apr2011	4	2011
4.	11feb2012	2	2012
5.	01aug2012	8	2012

Now, for each year, we can look at the patients that were admitted in the first three months and the reason for their visit:

```
. bysort adyear: list patid reason if admonth < 4
```

```
-> adyear = 2011
```

	patid	reason
2.	2	chest pain

```
-> adyear = 2012
```

	patid	reason
1.	4	abdominal pain

Example 3: Building dates from components

If we are concerned only with the month and year of admission, we can also create a monthly date with the two newly created variables above:

```
. generate monthly = ym(adyear,admonth)
. format monthly %tm
. list admit monthly
```

	admit	monthly
1.	25jun2011	2011m6
2.	13mar2011	2011m3
3.	09apr2011	2011m4
4.	11feb2012	2012m2
5.	01aug2012	2012m8

Because we now have monthly dates, we apply the `%tm` display format.

The ym() function shown above is useful when you have components of a date stored separately. In fact, we could have created this monthly date variable by nesting functions:

```
. generate monthly2 = ym(year(admit), month(admit))
. format monthly2 %tm
```

Instead of generating those intermediary variables to extract the month and year of the daily date, we simply used the extraction functions year() and month() within the ym() function. Either of the two methods shown above will give you the same result, but if your goal is to convert a daily date variable to a monthly date, you can use the mofd() conversion function, as demonstrated in the next example.

Example 4: Converting among date types

Often, we need to modify the data from its raw form for our purposes. For example, suppose our dataset included only the datetime variable admit_time but we were interested only in the date. We could type

```
. generate dateoftime = dofc(admit_time)
. format dateoftime %td
. list admit_time dateoftime
```

	admit_time	dateoft~e
1.	25jun2011 05:15:06	25jun2011
2.	13mar2011 08:30:45	13mar2011
3.	09apr2011 10:17:08	09apr2011
4.	11feb2012 10:30:12	11feb2012
5.	01aug2012 06:45:59	01aug2012

Or we might want to create a monthly date from the date of admission:

```
. generate monthofdate = mofd(admit)
. format monthofdate %tm
. list admit monthofdate
```

	admit	montho~e
1.	25jun2011	2011m6
2.	13mar2011	2011m3
3.	09apr2011	2011m4
4.	11feb2012	2012m2
5.	01aug2012	2012m8

Several functions are available for converting from one type of date and time to another. But, if one is not available for what you need, you can nest functions to obtain the conversion you want. For example, suppose we would like to convert a monthly date to a quarterly date. There is no direct function for this conversion, so instead we type

```
. generate quarterly = qofd(dofm(monthofdate))
. format quarterly %tq
. list monthofdate quarterly
```

	montho~e	quarte~y
1.	2011m6	2011q2
2.	2011m3	2011q1
3.	2011m4	2011q2
4.	2012m2	2012q1
5.	2012m8	2012q3

We use the `dofm()` function to convert the monthly date to a daily date. This daily date will contain the month and year from the monthly date, and the day will be set to 1. This is the general rule with datetime functions; if you are converting from one type of date to another that has more elements, those elements are set to their defaults. The `qofd()` function then converts the resulting daily date to a quarterly date.

Example 5: Using dates in expressions

Besides generating date and time variables, you might use dates in expressions. For example, suppose we wanted to look only at observations after a certain date. Let's list visit information for any patients who were admitted after February 20, 2012:

```
. list admit patid reason if admit > td(20feb2012)
```

	admit	patid	reason
5.	01aug2012	5	rapid breathing

This `td()` function will convert February 20, 2012, to its numeric form. Our expression is then evaluated by comparing this numeric value with the numeric values stored in `admit`.

If you would like to see that underlying numeric value, you can type

```
. display td(20feb2012)
```

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.

Cox, N. J., and C. B. Schechter. 2018. Speaking Stata: Seven steps for vexatious string variables. *Stata Journal* 18: 981–994.

Also see

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

[D] **Datetime conversion** — Converting strings to Stata dates

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

[D] **Datetime durations** — Obtaining and working with durations

[D] **Datetime relative dates** — Obtaining dates and date information from other dates

[D] **Datetime values from other software** — Date and time conversion from other software