Title

tsset — Declare data to be time-series data

Syntax Menu Description Options
Remarks and examples Stored results References Also see

Syntax

Declare data to be time series

`tsset` *timevar* [, *options*]

`tsset` *panelvar* *timevar* [, *options*]

Display how data are currently tsset

`tsset`

Clear time-series settings

`tsset, clear`

In the declare syntax, *panelvar* identifies the panels and *timevar* identifies the times.

options	Description
Main	
unitoptions	specify units of *timevar*
Delta	
deltaoption	specify period of *timevar*
`noquery`	suppress summary calculations and output

`noquery` is not shown in the dialog box.

unitoptions	Description
(default)	*timevar*'s units to be obtained from *timevar*'s display format
`clocktime`	*timevar* is `%tc`: 0 = 1jan1960 00:00:00.000, 1 = 1jan1960 00:00:00.001, ...
`daily`	*timevar* is `%td`: 0 = 1jan1960, 1 = 2jan1960, ...
`weekly`	*timevar* is `%tw`: 0 = 1960w1, 1 = 1960w2, ...
`monthly`	*timevar* is `%tm`: 0 = 1960m1, 1 = 1960m2, ...
`quarterly`	*timevar* is `%tq`: 0 = 1960q1, 1 = 1960q2,...
`halfyearly`	*timevar* is `%th`: 0 = 1960h1, 1 = 1960h2,...
`yearly`	*timevar* is `%ty`: 1960 = 1960, 1961 = 1961, ...
`generic`	*timevar* is `%tg`: 0 = ?, 1 = ?, ...
`format(%`*fmt*`)`	specify *timevar*'s format and then apply default rule

In all cases, negative *timevar* values are allowed.

deltaoption specifies the period between observations in *timevar* units and may be specified as

deltaoption	Example
`delta(#)`	`delta(1)` or `delta(2)`
`delta((`*exp*`))`	`delta((7*24))`
`delta(`*# units*`)`	`delta(7 days)` or `delta(15 minutes)` or `delta(7 days 15 minutes)`
`delta((`*exp*`)` *units*`)`	`delta((2+3) weeks)`

Allowed units for `%tc` and `%tC` *timevars* are

`seconds`	`second`	`secs`	`sec`
`minutes`	`minute`	`mins`	`min`
`hours`	`hour`		
`days`	`day`		
`weeks`	`week`		

and for all other `%t` *timevars*, units specified must match the frequency of the data; for example, for `%ty`, units must be year or years.

Menu

Statistics > Time series > Setup and utilities > Declare dataset to be time-series data

Description

`tsset` declares the data in memory to be a time series. `tsset`ting the data is what makes Stata's time-series operators such as `L.` and `F.` (lag and lead) work; the operators are discussed under *Remarks and examples* below. Also, before using the other `ts` commands, you must `tsset` the data first. If you `save` the data after `tsset`, the data will be remembered to be time series and you will not have to `tsset` again.

There are two syntaxes for setting the data:

`tsset` *timevar*
`tsset` *panelvar timevar*

In the first syntax—`tsset` *timevar*—the data are set to be a straight time series.

In the second syntax—`tsset` *panelvar timevar*—the data are set to be a collection of time series, one for each value of *panelvar*, also known as panel data, cross-sectional time-series data, and `xt` data. Such datasets can be analyzed by `xt` commands as well as `ts` commands. If you `tsset` *panelvar timevar*, you do not need to `xtset` *panelvar timevar* to use the `xt` commands.

`tsset` without arguments—`tsset`—displays how the data are currently `tsset` and sorts the data on *timevar* or *panelvar timevar* if they are sorted differently from that.

`tsset, clear` is a rarely used programmer's command to declare that the data are no longer a time series.

Options

Main

unitoptions `clocktime`, `daily`, `weekly`, `monthly`, `quarterly`, `halfyearly`, `yearly`, `generic`, and `format(`*%fmt*`)` specify the units in which *timevar* is recorded.

timevar will usually be a `%t` variable; see [D] **datetime**. If *timevar* already has a `%t` display format assigned to it, you do not need to specify a *unitoption*; `tsset` will obtain the units from the format. If you have not yet bothered to assign the appropriate `%t` format, however, you can use the *unitoptions* to tell `tsset` the units. Then `tsset` will set *timevar*'s display format for you. Thus, the *unitoptions* are convenience options; they allow you to skip formatting the time variable. The following all have the same net result:

Alternative 1	Alternative 2	Alternative 3
`format t %td`	*(t not formatted)*	*(t not formatted)*
`tsset t`	`tsset t, daily`	`tsset t, format(%td)`

timevar is not required to be a `%t` variable; it can be any variable of your own concocting so long as it takes on only integer values. In such cases, it is called generic and considered to be `%tg`. Specifying the *unitoption* `generic` or attaching a special format to *timevar*, however, is not necessary because `tsset` will assume that the variable is generic if it has any numerical format other than a `%t` format (or if it has a `%tg` format).

`clear`—used in `tsset, clear`—makes Stata forget that the data ever were `tsset`. This is a rarely used programmer's option.

Delta

`delta()` specifies the period of *timevar* and is commonly used when *timevar* is `%tc`. `delta()` is only sometimes used with the other `%t` formats or with generic time variables.

If `delta()` is not specified, `delta(1)` is assumed. This means that at *timevar* = 5, the previous time is *timevar* = 5 − 1 = 4 and the next time would be *timevar* = 5 + 1 = 6. Lag and lead operators, for instance, would work this way. This would be assumed regardless of the units of *timevar*.

If you specified `delta(2)`, then at *timevar* = 5, the previous time would be *timevar* = 5 − 2 = 3 and the next time would be *timevar* = 5 + 2 = 7. Lag and lead operators would work this way. In the observation with *timevar* = 5, `L.price` would be the value of `price` in the observation for which *timevar* = 3 and `F.price` would be the value of `price` in the observation for which *timevar* = 7. If you then add an observation with *timevar* = 4, the operators will still work appropriately; that is, at *timevar* = 5, `L.price` will still have the value of `price` at *timevar* = 3.

There are two aspects of *timevar*: its units and its periodicity. The *unitoptions* set the units. `delta()` sets the periodicity.

We mentioned that `delta()` is commonly used with `%tc` *timevars* because Stata's `%tc` variables have units of milliseconds. If `delta()` is not specified and in some model you refer to `L.price`, you will be referring to the value of `price` 1 ms ago. Few people have data with periodicity of a millisecond. Perhaps your data are hourly. You could specify `delta(3600000)`. Or you could specify `delta((60*60*1000))`, because `delta()` will allow expressions if you include an extra pair of parentheses. Or you could specify `delta(1 hour)`. They all mean the same thing: *timevar* has periodicity of 3,600,000 ms. In an observation for which *timevar* = 1,489,572,000,000 (corresponding to 15mar2007 10:00:00), `L.price` would be the observation for which *timevar* = 1,489,572,000,000 − 3,600,000 = 1,489,568,400,000 (corresponding to 15mar2007 9:00:00).

When you tsset the data and specify delta(), tsset verifies that all the observations follow the specified periodicity. For instance, if you specified delta(2), then *timevar* could contain any subset of $\{\ldots,-4,-2,0,2,4,\ldots\}$ or it could contain any subset of $\{\ldots,-3,-1,1,3,\ldots\}$. If *timevar* contained a mix of values, tsset would issue an error message. If you also specify a *panelvar*—you type tsset *panelvar timevar*, delta(2)—the check is made on each panel independently. One panel might contain *timevar* values from one set and the next, another, and that would be fine.

The following option is available with tsset but is not shown in the dialog box:

noquery prevents tsset from performing most of its summary calculations and suppresses output. With this option, only the following results are posted:

r(tdelta)	r(tsfmt)
r(panelvar)	r(unit)
r(timevar)	r(unit1)

Remarks and examples

stata.com

Remarks are presented under the following headings:

Overview

tsset sets *timevar* so that Stata's time-series operators are understood in varlists and expressions. The time-series operators are

Operator	Meaning
L.	lag x_{t-1}
L2.	2-period lag x_{t-2}
...	
F.	lead x_{t+1}
F2.	2-period lead x_{t+2}
...	
D.	difference $x_t - x_{t-1}$
D2.	difference of difference $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$
...	
S.	"seasonal" difference $x_t - x_{t-1}$
S2.	lag-2 (seasonal) difference $x_t - x_{t-2}$
...	

Time-series operators may be repeated and combined. L3.gnp refers to the third lag of variable gnp, as do LLL.gnp, LL2.gnp, and L2L.gnp. LF.gnp is the same as gnp. DS12.gnp refers to the one-period difference of the 12-period difference. LDS12.gnp refers to the same concept, lagged once.

D1. = S1., but D2. $\neq$ S2., D3. $\neq$ S3., and so on. D2. refers to the difference of the difference. S2. refers to the two-period difference. If you wanted the difference of the difference of the 12-period difference of gnp, you would write D2S12.gnp.

Operators may be typed in uppercase or lowercase. Most users would type `d2s12.gnp` instead of `D2S12.gnp`.

You may type operators however you wish; Stata internally converts operators to their canonical form. If you typed `ld2ls12d.gnp`, Stata would present the operated variable as `L2D3S12.gnp`.

Stata also understands *operator*`(`*numlist*`).` to mean a set of operated variables. For instance, typing `L(1/3).gnp` in a varlist is the same as typing '`L.gnp L2.gnp L3.gnp`'. The operators can also be applied to a list of variables by enclosing the variables in parentheses; for example,

```
. list year L(1/3).(gnp cpi)
```

	year	L.gnp	L2.gnp	L3.gnp	L.cpi	L2.cpi	L3.cpi
1.	1989	.	.	.	.	.	.
2.	1990	5452.8	.	.	100	.	.
3.	1991	5764.9	5452.8	.	105	100	.
4.	1992	5932.4	5764.9	5452.8	108	105	100
			(output omitted)				
8.	1996	7330.1	6892.2	6519.1	122	119	112

In *operator#*`.`, making # zero returns the variable itself. `L0.gnp` is `gnp`. Thus, you can type `list year l(0/3).gnp` to mean `list year gnp L.gnp L2.gnp L3.gnp`.

The parenthetical notation may be used with any operator. Typing `D(1/3).gnp` would return the first through third differences.

The parenthetical notation may be used in operator lists with multiple operators, such as `L(0/3)D2S12.gnp`.

Operator lists may include up to one set of parentheses, and the parentheses may enclose a *numlist*; see **[U] 11.1.8 numlist**.

Before you can use these time-series operators, however, the dataset must satisfy two requirements:

1. the dataset must be `tsset` and
2. the dataset must be sorted by *timevar* or, if it is a cross-sectional time-series dataset, by *panelvar timevar*.

`tsset` handles both requirements. As you use Stata, however, you may later use a command that re-sorts that data, and if you do, the time-series operators will not work:

```
. tsset time
  (output omitted)
. regress y x l.x
  (output omitted)
. (you continue to use Stata and, sometime later:)
. regress y x l.x
not sorted
r(5);
```

Then typing `tsset` without arguments will reestablish the sort order:

```
. tsset
  (output omitted)
. regress y x l.x
  (output omitted)
```

Here typing `tsset` is the same as typing `sort time`. Had we previously `tsset country time`, however, typing `tsset` would be the same as typing `sort country time`. You can type the `sort` command or type `tsset` without arguments; it makes no difference.

There are two syntaxes for setting your data:

```
tsset timevar
tsset panelvar timevar
```

In both, *timevar* must contain integer values. If *panelvar* is specified, it too must contain integer values, and the dataset is declared to be a cross-section of time series, such as a collection of time series for different countries.

▷ Example 1: Numeric time variable

You have monthly data on personal income. Variable `t` records the time of an observation, but there is nothing special about the name of the variable. There is nothing special about the values of the variable, either. `t` is not required to be `%tm` variable—perhaps you do not even know what that means. `t` is just a numeric variable containing integer values that represent the month, and we will imagine that `t` takes on the values 1, 2, ..., 9, although it could just as well be −3, −2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the values are dense: adjacent months have a time value that differs by 1.

```
. use http://www.stata-press.com/data/r13/tssetxmpl
. list t income

     +------------+
     | t   income |
     |------------|
  1. | 1     1153 |
  2. | 2     1181 |
    (output omitted)
  9. | 9     1282 |
     +------------+

. tsset t
        time variable:  t, 1 to 9
                delta:  1 unit
. regress income l.income
  (output omitted)
```

◁

▷ Example 2: Adjusting the starting date

In the example above, that `t` started at 1 was not important. As we said, the `t` variable could just as well be recorded −3, −2 ..., 5, or 1,023, 1,024, ..., 1,031. What is important is that the difference in `t` between observations be `delta()` when there are no gaps.

Although how time is measured makes no difference, Stata has formats to display time nicely if it is recorded in certain ways; you can learn about the formats by seeing [D] **datetime**. Stata likes time variables in which 1jan1960 is recorded as 0. In our previous example, if `t` = 1 corresponds to July 1995, then we could make a variable that fits Stata's preference by typing

```
. generate newt = tm(1995m7) + t - 1
```

`tm()` is the function that returns a month equivalent; `tm(1995m7)` evaluates to the constant 426, meaning 426 months after January 1960. We now have variable `newt` containing

```
. list t newt income
```

	t	newt	income
1.	1	426	1153
2.	2	427	1181
3.	3	428	1208
		(output omitted)	
9.	9	434	1282

If we put a `%tm` format on `newt`, it will display more cleanly:

```
. format newt %tm
. list t newt income
```

	t	newt	income
1.	1	1995m7	1153
2.	2	1995m8	1181
3.	3	1995m9	1208
		(output omitted)	
9.	9	1996m3	1282

We could now `tsset newt` rather than `t`:

```
. tsset newt
        time variable:  newt, 1995m7 to 1996m3
                delta:  1 month
```

◁

❑ Technical note

In addition to monthly, Stata understands clock times (to the millisecond level) as well as daily, weekly, quarterly, half-yearly, and yearly data. See [D] **datetime** for a description of these capabilities.

Let's reconsider the previous example, but rather than monthly, let's assume the data are daily, weekly, etc. The only thing to know is that, corresponding to function `tm()`, there are functions `td()`, `tw()`, `tq()`, `th()`, and `ty()` and that, corresponding to format `%tm`, there are formats `%td`, `%tw`, `%tq`, `%th`, and `%ty`. Here is what we would have typed had our data been on a different time scale:

Daily: if your `t` variable had `t=1` corresponding to 15mar1993

```
. gen newt = td(15mar1993) + t - 1
. tsset newt, daily
```

Weekly: if your `t` variable had `t=1` corresponding to 1994w1:

```
. gen newt = tw(1994w1) + t - 1
. tsset newt, weekly
```

Monthly: if your `t` variable had `t=1` corresponding to 2004m7:

```
. gen newt = tm(2004m7) + t - 1
. tsset newt, monthly
```

Quarterly: if your `t` variable had `t=1` corresponding to 1994q1:

```
. gen newt = tq(1994q1) + t - 1
. tsset newt, quarterly
```

Half-yearly: if your `t` variable had `t=1` corresponding to 1921h2:

```
. gen newt = th(1921h2) + t - 1
. tsset newt, halfyearly
```

Yearly: if your `t` variable had `t=1` corresponding to 1842:

```
. gen newt = 1842 + t - 1
. tsset newt, yearly
```

In each example above, we subtracted one from our time variable in constructing the new time variable `newt` because we assumed that our starting time value was 1. For the quarterly example, if our starting time value were 5 and that corresponded to `1994q1`, we would type

```
. generate newt = tq(1994q1) + t - 5
```

Had our initial time value been `t` = 742 and that corresponded to `1994q1`, we would have typed

```
. generate newt = tq(1994q1) + t - 742
```

❑

▷ Example 3: Time-series data but no time variable

Perhaps we have the same time-series data but no time variable:

```
. use http://www.stata-press.com/data/r13/tssetxmpl2, clear
. list income

     +--------+
     | income |
     |--------|
  1. |   1153 |
  2. |   1181 |
  3. |   1208 |
  4. |   1272 |
  5. |   1236 |
     |--------|
  6. |   1297 |
  7. |   1265 |
  8. |   1230 |
  9. |   1282 |
     +--------+
```

Say that we know that the first observation corresponds to July 1995 and continues without gaps. We can create a monthly time variable and format it by typing

```
. generate t = tm(1995m7) + _n - 1
. format t %tm
```

We can now `tsset` our dataset and `list` it:

```
. tsset t
        time variable:  t, 1995m7 to 1996m3
                delta:  1 month
. list t income

     +------------------+
     |      t    income |
     |------------------|
  1. | 1995m7      1153 |
  2. | 1995m8      1181 |
  3. | 1995m9      1208 |
        (output omitted)
  9. | 1996m3      1282 |
     +------------------+
```

◁

▷ Example 4: Time variable as a string

Your data might include a time variable that is encoded into a string. In the example below each monthly observation is identified by string variable `yrmo` containing the month and year of the observation, sometimes with punctuation between:

```
. use http://www.stata-press.com/data/r13/tssetxmpl, clear
. list yrmo income
```

	yrmo	income
1.	7/1995	1153
2.	8/1995	1181
3.	9-1995	1208
4.	10,1995	1272
5.	11 1995	1236
6.	12 1995	1297
7.	1/1996	1265
8.	2.1996	1230
9.	3- 1996	1282

The first step is to convert the string to a numeric representation. Doing so is easy using the `monthly()` function; see [D] **datetime**.

```
. gen mdate = monthly(yrmo, "MY")
. list yrmo mdate income
```

	yrmo	mdate	income
1.	7/1995	426	1153
2.	8/1995	427	1181
3.	9-1995	428	1208
	(output omitted)		
9.	3- 1996	434	1282

Our new variable, `mdate`, contains the number of months from January 1960. Now that we have numeric variable `mdate`, we can `tsset` the data:

```
. format mdate %tm
. tsset mdate
        time variable:  mdate, 1995m7 to 1996m3
                delta:  1 month
```

In fact, we can combine the two and type

```
. tsset mdate, format(%tm)
        time variable:  mdate, 1995m7 to 1996m3
                delta:  1 month
```

or type

```
. tsset mdate, monthly
        time variable:  mdate, 1995m7 to 1996m3
                delta:  1 month
```

In all cases, we obtain

```
. list yrmo mdate income

     +----------------------------+
     |    yrmo     mdate   income |
     |----------------------------|
  1. |  7/1995    1995m7     1153 |
  2. |  8/1995    1995m8     1181 |
  3. |  9-1995    1995m9     1208 |
  4. | 10,1995   1995m10     1272 |
  5. | 11 1995   1995m11     1236 |
     |----------------------------|
  6. | 12 1995   1995m12     1297 |
  7. |  1/1996    1996m1     1265 |
  8. |  2.1996    1996m2     1230 |
  9. | 3- 1996    1996m3     1282 |
     +----------------------------+
```

Stata can translate many different date formats, including strings like 12jan2009; January 12, 2009; 12-01-2009; 01/12/2009; 01/12/09; 12jan2009 8:14; 12-01-2009 13:12; 01/12/09 1:12 pm; Wed Jan 31 13:03:25 CST 2009; 1998q1; and more. See [D] **datetime**.

◁

▷ Example 5: Time-series data with gaps

Gaps in the time series cause no difficulties:

```
. use http://www.stata-press.com/data/r13/tssetxmpl3, clear

. list yrmo income

     +------------------+
     |    yrmo   income |
     |------------------|
  1. |  7/1995     1153 |
  2. |  8/1995     1181 |
  3. | 11 1995     1236 |
  4. | 12 1995     1297 |
  5. |  1/1996     1265 |
     |------------------|
  6. | 3- 1996     1282 |
     +------------------+

. gen mdate = monthly(yrmo, "MY")

. tsset mdate, monthly
        time variable:  mdate, 1995m7 to 1996m3, but with gaps
                delta:  1 month
```

Once the dataset has been `tsset`, we can use the time-series operators. The `D` operator specifies first differences:

```
. list mdate income d.income

     +-----------------------------+
     |   mdate   income   D.income |
     |-----------------------------|
  1. |  1995m7     1153          . |
  2. |  1995m8     1181         28 |
  3. | 1995m11     1236          . |
  4. | 1995m12     1297         61 |
  5. |  1996m1     1265        -32 |
     |-----------------------------|
  6. |  1996m3     1282          . |
     +-----------------------------+
```

We can use the operators in an expression or varlist context; we do not have to create a new variable to hold `D.income`. We can use `D.income` with the `list` command, with `regress` or any other Stata command that allows time-series varlists.

◁

▷ Example 6: Clock times

We have data from a large hotel in Las Vegas that changes the reservation prices for its rooms hourly. A piece of the data looks like

```
. use http://www.stata-press.com/data/r13/tssetxmpl4, clear
. list in 1/5
```

	time	price
1.	02.13.2007 08:00	140
2.	02.13.2007 09:00	155
3.	02.13.2007 10:00	160
4.	02.13.2007 11:00	155
5.	02.13.2007 12:00	160

Variable `time` is a string variable. The first step in making this dataset a time-series dataset is to translate the string to a numeric variable:

```
. generate double t = clock(time, "MDY hm")
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1.487e+12
2.	02.13.2007 09:00	155	1.487e+12
3.	02.13.2007 10:00	160	1.487e+12
4.	02.13.2007 11:00	155	1.487e+12
5.	02.13.2007 12:00	160	1.487e+12

See [D] **datetime** for an explanation of what is going on here. `clock()` is the function that converts strings to datetime (`%tc`) values. We typed `clock(time, "MDY hm")` to convert string variable `time`, and we told `clock()` that the values in `time` were in the order month, day, year, hour, and minute. We stored new variable `t` as a `double` because time values are large, and doing so is required to prevent rounding. Even so, the resulting values 1.487e+12 look rounded, but that is only because of the default display format for new variables. We can see the values better if we change the format:

```
. format t %20.0gc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1,486,972,800,000
2.	02.13.2007 09:00	155	1,486,976,400,000
3.	02.13.2007 10:00	160	1,486,980,000,000
4.	02.13.2007 11:00	155	1,486,983,600,000
5.	02.13.2007 12:00	160	1,486,987,200,000

Even better would be to change the format to %tc—Stata's clock-time format:

```
. format t %tc
. list in 1/5

     +------------------------------------------------+
     |             time   price                     t |
     |------------------------------------------------|
  1. | 02.13.2007 08:00     140    13feb2007 08:00:00 |
  2. | 02.13.2007 09:00     155    13feb2007 09:00:00 |
  3. | 02.13.2007 10:00     160    13feb2007 10:00:00 |
  4. | 02.13.2007 11:00     155    13feb2007 11:00:00 |
  5. | 02.13.2007 12:00     160    13feb2007 12:00:00 |
     +------------------------------------------------+
```

We could drop variable `time`. New variable `t` contains the same information as `time` and `t` is better because it is a Stata time variable, the most important property of which being that it is numeric rather than string. We can `tsset` it. Here, however, we also need to specify the period with `tsset`'s `delta()` option. Stata's time variables are numeric, but they record milliseconds since 01jan1960 00:00:00. By default, `tsset` uses `delta(1)`, and that means the time-series operators would not work as we want them to work. For instance, `L.price` would look back only 1 ms (and find nothing). We want `L.price` to look back 1 hour (3,600,000 ms):

```
. tsset t, delta(1 hour)
        time variable:  t,
                        13feb2007 08:00:00.000 to 13feb2007 14:00:00.000
                delta:  1 hour
. list t price l.price in 1/5

     +---------------------------------------+
     |                  t   price   L.price  |
     |---------------------------------------|
  1. | 13feb2007 08:00:00     140         .  |
  2. | 13feb2007 09:00:00     155       140  |
  3. | 13feb2007 10:00:00     160       155  |
  4. | 13feb2007 11:00:00     155       160  |
  5. | 13feb2007 12:00:00     160       155  |
     +---------------------------------------+
```

◁

▷ Example 7: Clock times must be double

In the previous example, it was of vital importance that when we generated the %tc variable `t`,

```
. generate double t = clock(time, "MDY hm")
```

we generated it as a `double`. Let's see what would have happened had we forgotten and just typed `generate t = clock(time, "MDY hm")`. Let's go back and start with the same original data:

```
. use http://www.stata-press.com/data/r13/tssetxmpl4, clear
. list in 1/5

     +--------------------------+
     |             time   price |
     |--------------------------|
  1. | 02.13.2007 08:00     140 |
  2. | 02.13.2007 09:00     155 |
  3. | 02.13.2007 10:00     160 |
  4. | 02.13.2007 11:00     155 |
  5. | 02.13.2007 12:00     160 |
     +--------------------------+
```

Remember, variable `time` is a string variable, and we need to translate it to numeric. So we translate, but this time we forget to make the new variable a `double`:

```
. generate t = clock(time, "MDY hm")
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1.49e+12
2.	02.13.2007 09:00	155	1.49e+12
3.	02.13.2007 10:00	160	1.49e+12
4.	02.13.2007 11:00	155	1.49e+12
5.	02.13.2007 12:00	160	1.49e+12

We see the first difference—`t` now lists as 1.49e+12 rather than 1.487e+12 as it did previously—but this is nothing that would catch our attention. We would not even know that the value is different. Let's continue.

We next put a `%20.0gc` format on `t` to better see the numerical values. In fact, that is not something we would usually do in an analysis. We did that in the example to emphasize to you that the `t` values were really big numbers. We will repeat the exercise just to be complete, but in real analysis, we would not bother.

```
. format t %20.0gc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	1,486,972,780,544
2.	02.13.2007 09:00	155	1,486,976,450,560
3.	02.13.2007 10:00	160	1,486,979,989,504
4.	02.13.2007 11:00	155	1,486,983,659,520
5.	02.13.2007 12:00	160	1,486,987,198,464

Okay, we see big numbers in `t`. Let's continue.

Next we put a `%tc` format on `t`, and that is something we would usually do, and you should always do. You should also list a bit of the data, as we did:

```
. format t %tc
. list in 1/5
```

	time	price	t
1.	02.13.2007 08:00	140	13feb2007 07:59:40
2.	02.13.2007 09:00	155	13feb2007 09:00:50
3.	02.13.2007 10:00	160	13feb2007 09:59:49
4.	02.13.2007 11:00	155	13feb2007 11:00:59
5.	02.13.2007 12:00	160	13feb2007 11:59:58

By now, you should see a problem: the translated datetime values are off by a second or two. That was caused by rounding. Dates and times should be the same, not approximately the same, and when you see a difference like this, you should say to yourself, "The translation is off a little. Why is that?" and then you should think, "Of course, rounding. I bet that I did not create `t` as a `double`."

Let us assume, however, that you do not do this. You instead plow ahead:

```
. tsset t, delta(1 hour)
time values with period less than delta() found
r(451);
```

And that is what will happen when you forget to create `t` as a `double`. The rounding will cause uneven period, and `tsset` will complain.

By the way, it is only important that clock times (`%tc` and `%tC` variables) be stored as `doubles`. The other date values `%td`, `%tw`, `%tm`, `%tq`, `%th`, and `%ty` are small enough that they can safely be stored as `floats`, although forgetting and storing them as `doubles` does no harm.

◁

❏ Technical note

Stata provides two clock-time formats, `%tc` and `%tC`. `%tC` provides a clock with leap seconds. Leap seconds are occasionally inserted to account for randomness of the earth's rotation, which gradually slows. Unlike the extra day inserted in leap years, the timing of when leap seconds will be inserted cannot be foretold. The authorities in charge of such matters announce a leap second approximately 6 months before insertion. Leap seconds are inserted at the end of the day, and the leap second is called 23:59:60 (that is, 11:59:60 pm), which is then followed by the usual 00:00:00 (12:00:00 am). Most nonastronomers find these leap seconds vexing. The added seconds cause problems because of their lack of predictability—knowing how many seconds there will be between 01jan2012 and 01jan2013 is not possible—and because there are not necessarily 24 hours in a day. If you use a leap second adjusted–clock, most days have 24 hours, but a few have 24 hours and 1 second. You must look at a table to find out.

From a time-series analysis point of view, the nonconstant day causes the most problems. Let's say that you have data on blood pressure, taken hourly at 1:00, 2:00, . . . , and that you have `tsset` your data with `delta(1 hour)`. On most days, `L24.bp` would be blood pressure at the same time yesterday. If the previous day had a leap second, however, and your data were recorded using a leap second adjusted–clock, there would be no observation `L24.bp` because 86,400 seconds before the current reading does not correspond to an on-the-hour time; 86,401 seconds before the current reading corresponds to yesterday's time. Thus, whenever possible, using Stata's `%tc` encoding rather than `%tC` is better.

When times are recorded by computers using leap second–adjusted clocks, however, avoiding `%tC` is not possible. For performing most time-series analysis, the recommended procedure is to map the `%tC` values to `%tc` and then `tsset` those. You must ask yourself whether the process you are studying is based on the clock—the nurse does something at 2 o'clock every day—or the true passage of time—the emitter spits out an electron every 86,400,000 ms.

When dealing with computer-recorded times, first find out whether the computer (and its time-recording software) use a leap second–adjusted clock. If it does, translate that to a `%tC` value. Then use function `cofC()` to convert to a `%tc` value and `tsset` that. If variable `T` contains the `%tC` value,

```
. gen double t = cofC(T)
. format t %tc
. tsset t, delta(...)
```

Function `cofC()` moves leap seconds forward: 23:59:60 becomes 00:00:00 of the next day.

❏

Panel data

▷ Example 8: Time-series data for multiple groups

Assume that we have a time series on average annual income and that we have the series for two groups: individuals who have not completed high school (`edlevel` = 1) and individuals who have (`edlevel` = 2).

```
. use http://www.stata-press.com/data/r13/tssetxmpl5, clear
. list edlevel year income, sep(0)
```

	edlevel	year	income
1.	1	1988	14500
2.	1	1989	14750
3.	1	1990	14950
4.	1	1991	15100
5.	2	1989	22100
6.	2	1990	22200
7.	2	1992	22800

We declare the data to be a panel by typing

```
. tsset edlevel year, yearly
       panel variable:  edlevel, (unbalanced)
        time variable:  year, 1988 to 1992, but with a gap
                delta:  1 year
```

Having `tsset` the data, we can now use time-series operators. The difference operator, for example, can be used to list annual changes in income:

```
. list edlevel year income d.income, sep(0)
```

	edlevel	year	income	D.income
1.	1	1988	14500	.
2.	1	1989	14750	250
3.	1	1990	14950	200
4.	1	1991	15100	150
5.	2	1989	22100	.
6.	2	1990	22200	100
7.	2	1992	22800	.

We see that in addition to producing missing values due to missing times, the difference operator correctly produced a missing value at the start of each panel. Once we have `tsset` our panel data, we can use time-series operators and be assured that they will handle missing time periods and panel changes correctly.

◁

Video example

Time series, part 1: Formatting dates, tsset, tsreport, and tsfill

Stored results

`tsset` stores the following in `r()`:

Scalars

`r(imin)`	minimum panel ID
`r(imax)`	maximum panel ID
`r(tmin)`	minimum time
`r(tmax)`	maximum time
`r(tdelta)`	delta

Macros

`r(panelvar)`	name of panel variable
`r(timevar)`	name of time variable
`r(tdeltas)`	formatted delta
`r(tmins)`	formatted minimum time
`r(tmaxs)`	formatted maximum time
`r(tsfmt)`	*%fmt* of time variable
`r(unit)`	units of time variable: `Clock`, `clock`, `daily`, `weekly`, `monthly`, `quarterly`, `halfyearly`, `yearly`, or `generic`
`r(unit1)`	units of time variable: `C`, `c`, `d`, `w`, `m`, `q`, `h`, `y`, or `""`
`r(balanced)`	`unbalanced`, `weakly balanced`, or `strongly balanced`; a set of panels are strongly balanced if they all have the same time values, otherwise balanced if same number of time values, otherwise unbalanced

References

Baum, C. F. 2000. sts17: Compacting time series data. *Stata Technical Bulletin* 57: 44–45. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 369–370. College Station, TX: Stata Press.

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.

Also see

[TS] **tsfill** — Fill in gaps in time variable