Title

stata.com

xtset — Declare data to be panel data

Description

`xtset` manages the panel settings of a dataset. You must `xtset` your data before you can use the other `xt` commands. `xtset` *panelvar* declares the data in memory to be a panel in which the order of observations is irrelevant. `xtset` *panelvar timevar* declares the data to be a panel in which the order of observations is relevant. When you specify *timevar*, you can then use Stata's time-series operators and analyze your data with the `ts` commands without having to `tsset` your data.

`xtset` without arguments displays how the data are currently `xtset`. If the data are set with a *panelvar* and a *timevar*, `xtset` also sorts the data by *panelvar timevar* if a *timevar* was specified. If the data are set with a *panelvar* only, the sort order is not changed.

`xtset, clear` is a rarely used programmer's command to declare that the data are no longer to be considered a panel.

Quick start

Declare dataset to be panel data with panel identifier `pvar`

```
xtset pvar
```

Indicate that observations are ordered by year, stored in `tvar1`

```
xtset pvar tvar1
```

As above, but indicate that observations are instead made every 2 years

```
xtset pvar tvar1, delta(2)
```

Indicate that observations are made monthly; `tvar2` is not formatted

```
xtset pvar tvar2, monthly
```

As above, and apply `%tm` format to `tvar2`

```
xtset pvar tvar2, format(%tm)
```

Menu

Statistics > Longitudinal/panel data > Setup and utilities > Declare dataset to be panel data

Syntax

Declare data to be panel

```
xtset panelvar
xtset panelvar timevar [ , tsoptions ]
```

Display how data are currently xtset

```
xtset
```

Clear xt settings

```
xtset, clear
```

In the declare syntax, *panelvar* identifies the panels and the optional *timevar* identifies the times within panels. *tsoptions* concern *timevar*.

tsoptions	Description
unitoptions	specify units of *timevar*
deltaoption	specify length of period of *timevar*
`noquery`	suppress summary calculations and output

`noquery` is not shown in the dialog box.

unitoptions	Description
(default)	*timevar*'s units 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* are

```
days   day
weeks  week
```

Options

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 variable that counts 1, 2, ..., and is to be interpreted as first year of survey, second year, ..., or first month of treatment, second month, ... In these cases, you do not need to specify a *unitoption*.

In other cases, *timevar* will be a year variable or the like such as 2001, 2002, ..., and is to be interpreted as year of survey or the like. In those cases, you do not need to specify a *unitoption*.

In other, more complicated cases, *timevar* will be a full-blown %t variable; see [D] **datetime**. If *timevar* already has a %t display format assigned to it, you do not need to specify a *unitoption*; xtset will obtain the units from the format. If you have not yet bothered to assign the appropriate %t format to the %t variable, however, you can use the *unitoptions* to tell xtset the units. Then xtset 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)*
xtset pid t	xtset pid t, daily	xtset pid 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. When you xtset a time variable that is not %t, the display format does not change unless you specify the *unitoption* generic or use the format() option.

delta() specifies the period between observations in *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.income` would be the value of `income` in the observation for which *timevar* = 3 and `F.income` would be the value of `income` 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.income` will still have the value of `income` at *timevar* = 3.

There are two aspects of *timevar*: its units and its length of period. The *unitoptions* set the units. `delta()` sets the length of period. You are not required to specify one to specify the other. You might have a generic *timevar* but it counts in 12: 0, 12, 24, ... You would skip specifying *unitoptions* but would specify `delta(12)`.

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.bp`, you will be referring to the value of `bp` 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.bp` 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 `xtset` the data and specify `delta()`, `xtset` 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, `xtset` would issue an error message. The check is made on each panel independently, so one panel might contain *timevar* values from one set and the next, another, and that would be fine.

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

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

`noquery` prevents `xtset` 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

`xtset` declares the dataset in memory to be panel data. You need to do this before you can use the other `xt` commands. The storage types of both *panelvar* and *timevar* must be numeric, and both variables must contain integers only.

There are two syntaxes for setting the data:

`xtset` *panelvar*
`xtset` *panelvar timevar*

In the first syntax—`xtset` *panelvar*—the data are set to be a panel and the order of the observations within panel is considered to be irrelevant. For instance, *panelvar* might be country and the observations within might be city.

In the second syntax—`xtset` *panelvar timevar*—the data are to be a panel and the order of observations within panel are considered ordered by *timevar*. For instance, in data collected from repeated surveying of the same people over various years, *panelvar* might be person and *timevar*, year. When you specify *timevar*, you may then use Stata's time-series operators such as `L.` and `F.` (lag and lead) in other commands. The operators will be interpreted as lagged and lead values within panel.

The storage types of both *panelvar* and *timevar* must be numeric, and both variables must contain integers only.

❑ Technical note

In previous versions of Stata there was no `xtset` command. The other xt commands instead had the `i(`*panelvar*`)` and `t(`*timevar*`)` options. Older commands still have those options, but they are no longer documented and, if you specify them, they just perform the `xtset` for you. Thus, do-files that you previously wrote will continue to work. Modern usage, however, is to `xtset` the data first.

❑

❑ Technical note

`xtset` is related to the `tsset` command, which declares data to be time series. One of the syntaxes of `tsset` is `tsset` *panelvar timevar*, which is identical to one of `xtset`'s syntaxes, namely, `xtset` *panelvar timevar*. Here they are in fact the same command, meaning that `xtset`ting your data is sufficient to allow you to use the `ts` commands and `tsset`ting your data is sufficient to allow you to use the xt commands. You do not need to set both, but it will not matter if you do.

`xtset` and `tsset` are different, however, when you set just a *panelvar*—you type `xtset` *panelvar*—or when you set just a *timevar*—you type `tsset` *timevar*.

❑

If you `save` your data after `xtset`, the data will be remembered to be a panel and you will not have to `xtset` again.

▷ Example 1: Panel data without a time variable

Many panel datasets contain a variable identifying panels but do not contain a time variable. For example, you may have a dataset where each panel is a family, and the observations within panel are family members, or you may have a dataset in which each person made a decision multiple times but the ordering of those decisions is unimportant and perhaps unknown. In this latter case, if the time of the decision were known, we would advise you to `xtset` it. The other xt statistical commands do not do something different because *timevar* has been set—they will ignore *timevar* if *timevar* is irrelevant to the statistical method that you are using. You should always set everything that is true about the data.

In any case, let's consider the case where there is no *timevar*. We have data on U.S. states and cities within states:

```
. list state city in 1/10, sepby(state)
```

	state	city
1.	Alabama	Birmingham
2.	Alabama	Mobile
3.	Alabama	Montgomery
4.	Alabama	Huntsville
5.	Alaska	Anchorage
6.	Alaska	Fairbanks
7.	Arizona	Phoenix
8.	Arizona	Tucson
9.	Arkansas	Fayetteville
10.	Arkansas	Fort Smith

Here we do not type `xtset state city` because `city` is not a time variable. Instead, we type `xtset state`:

```
. xtset state
string variables not allowed in varlist;
state is a string variable
r(109);
```

You cannot `xtset` a string variable. We must make a numeric variable from our string variable and `xtset` that. One alternative is

```
. egen statenum = group(state)
. list state statenum in 1/10, sepby(state)
```

	state	statenum
1.	Alabama	1
2.	Alabama	1
3.	Alabama	1
4.	Alabama	1
5.	Alaska	2
6.	Alaska	2
7.	Arizona	3
8.	Arizona	3
9.	Arkansas	4
10.	Arkansas	4

```
. xtset statenum
       panel variable:  statenum (unbalanced)
```

Perhaps a better alternative is

```
. encode state, gen(st)
. list state st in 1/10, sepby(state)
```

	state	st
1.	Alabama	Alabama
2.	Alabama	Alabama
3.	Alabama	Alabama
4.	Alabama	Alabama
5.	Alaska	Alaska
6.	Alaska	Alaska
7.	Arizona	Arizona
8.	Arizona	Arizona
9.	Arkansas	Arkansas
10.	Arkansas	Arkansas

`encode` (see [D] **encode**) produces a numeric variable with a value label, so when we list the result, new variable `st` looks just like our original. It is, however, numeric:

```
. list state st in 1/10, nolabel sepby(state)
```

	state	st
1.	Alabama	1
2.	Alabama	1
3.	Alabama	1
4.	Alabama	1
5.	Alaska	2
6.	Alaska	2
7.	Arizona	3
8.	Arizona	3
9.	Arkansas	4
10.	Arkansas	4

We can `xtset` new variable `st`:

```
. xtset st
       panel variable:  st (unbalanced)
```

◁

Example 2: Panel data with a time variable

Some panel datasets do contain a time variable. Dataset `abdata.dta` contains labor demand data from a panel of firms in the United Kingdom. Here are wage data for the first two firms in the dataset:

```
. use http://www.stata-press.com/data/r15/abdata, clear
. list id year wage if id==1 | id==2, sepby(id)
```

	id	year	wage
1.	1	1977	13.1516
2.	1	1978	12.3018
3.	1	1979	12.8395
4.	1	1980	13.8039
5.	1	1981	14.2897
6.	1	1982	14.8681
7.	1	1983	13.7784
8.	2	1977	14.7909
9.	2	1978	14.1036
10.	2	1979	14.9534
11.	2	1980	15.491
12.	2	1981	16.1969
13.	2	1982	16.1314
14.	2	1983	16.3051

To declare this dataset as a panel dataset, you type

```
. xtset id year, yearly
       panel variable:  id (unbalanced)
        time variable:  year, 1976 to 1984
                delta:  1 year
```

The output from `list` shows that the last observations for these two firms are for 1983, but `xtset` shows that for some firms data are available for 1984 as well. If one or more panels contain data for nonconsecutive periods, `xtset` will report that gaps exist in the time variable. For example, if we did not have data for firm 1 for 1980 but did have data for 1979 and 1981, `xtset` would indicate that our data have a gap.

For yearly data, we could omit the `yearly` option and just type `xtset id year` because years are stored and listed just like regular integers.

Having declared our data to be a panel dataset, we can use time-series operators to obtain lags:

```
. list id year wage L.wage if id==1 | id==2, sepby(id)
```

	id	year	wage	L. wage
1.	1	1977	13.1516	.
2.	1	1978	12.3018	13.1516
3.	1	1979	12.8395	12.3018
4.	1	1980	13.8039	12.8395
5.	1	1981	14.2897	13.8039
6.	1	1982	14.8681	14.2897
7.	1	1983	13.7784	14.8681
8.	2	1977	14.7909	.
9.	2	1978	14.1036	14.7909
10.	2	1979	14.9534	14.1036
11.	2	1980	15.491	14.9534
12.	2	1981	16.1969	15.491
13.	2	1982	16.1314	16.1969
14.	2	1983	16.3051	16.1314

`L.wage` is missing for 1977 in both panels because we have no wage data for 1976. In observation 8, the lag operator did not incorrectly reach back into the previous panel.

◁

❑ Technical note

The terms *balanced* and *unbalanced* are often used to describe whether a panel dataset is missing some observations. If a dataset does not contain a time variable, then panels are considered *balanced* if each panel contains the same number of observations; otherwise, the panels are *unbalanced*.

When the dataset contains a time variable, panels are said to be *strongly balanced* if each panel contains the same time points, *weakly balanced* if each panel contains the same number of observations but not the same time points, and *unbalanced* otherwise.

❑

▷ Example 3: Applying time-series formats to the time variable

If our data are observed more than once per year, applying time-series formats to the time variable can improve readability.

We have a dataset consisting of individuals who joined a gym's weight-loss program that began in January 2005 and ended in December 2005. Each participant's weight was recorded once per month. Some participants did not show up for all the monthly weigh-ins, so we do not have all 12 months' records for each person. The first two people's data are

```
. use http://www.stata-press.com/data/r15/gymdata
. list id month wt if id==1 | id==2, sepby(id)
```

	id	month	wt
1.	1	1	145
2.	1	2	144
	(*output omitted*)		
11.	1	11	124
12.	1	12	120
13.	2	1	144
14.	2	2	143
	(*output omitted*)		
23.	2	11	122
24.	2	12	118

To set these data, we can type

```
. xtset id month
       panel variable:  id (unbalanced)
        time variable:  month, 1 to 12, but with gaps
                delta:  1 unit
```

The note "but with gaps" above is no cause for concern. It merely warns us that, within some panels, some time values are missing. We already knew that about our data—some participants did not show up for the monthly weigh-ins.

The rest of this example concerns making output more readable. Month numbers such as 1, 2, . . . , 12 are perfectly readable here. In another dataset, where month numbers went to, say 127, they would not be so readable. In such cases, we can make a more readable date—2005m1, 2005m2, . . . —by using Stata's `%t` variables. For a discussion, see [D] **datetime**. We will go quickly here. One of the `%t` formats is `%tm`—monthly—and it says that 1 means 1960m1. Thus, we need to recode our `month` variable so that, rather than taking on values from 1 to 12, it takes on values from 540 to 551. Then we can put a `%tm` format on that variable. Working out 540–551 is subject to mistakes. Stata function `tm(2005m1)` tells us the `%tm` month corresponding to January of 2005, so we can type

```
. generate month2 = month + m(2005m1) - 1
. format month2 %tm
```

New variable `month2` will work just as well as the original `month` in an `xtset`, and even a little better, because output will be a little more readable:

```
. xtset id month2
       panel variable:  id (unbalanced)
        time variable:  month2, 2005m1 to 2005m12, but with gaps
                delta:  1 month
```

By the way, we could have omitted typing `format month2 %tm` and then, rather than typing `xtset id month2`, we would have typed `xtset id month2, monthly`. The `monthly` option specifies that the time variable is `%tm`. When we did not specify the option, `xtset` determined that it was monthly from the display format we had set.

◁

Example 4: 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

```
. list in 1/5
```

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

The panel variable is `roomtype` and, although you cannot see it from the output above, it takes on 1, 2, . . . , 20. Variable `time` is a string variable. The first step in making this dataset xt is to translate the string to a numeric variable:

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

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

See [D] **datetime translation** 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 that 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
```

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

	roomtype	time	price	t
1.	1	02.13.2007 08:00	140	13feb2007 08:00:00
2.	1	02.13.2007 09:00	155	13feb2007 09:00:00
3.	1	02.13.2007 10:00	160	13feb2007 10:00:00
4.	1	02.13.2007 11:00	155	13feb2007 11:00:00
5.	1	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 xtset it. Here, however, we also need to specify the length of the periods with xtset's delta() option. Stata's time variables are numeric, but they record milliseconds since 01jan1960 00:00:00. By default, xtset 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):

```
. xtset roomtype t, delta(1 hour)
       panel variable:  roomtype (strongly balanced)
        time variable:  t, 13feb2007 08:00:00 to 31mar2007 18:00:00
                        but with gaps
                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 5: 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:

```
. list in 1/5
```

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

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

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

	roomtype	time	price	t
1.	1	02.13.2007 08:00	140	13feb2007 07:59:40
2.	1	02.13.2007 09:00	155	13feb2007 09:00:50
3.	1	02.13.2007 10:00	160	13feb2007 09:59:49
4.	1	02.13.2007 11:00	155	13feb2007 11:00:59
5.	1	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's assume, however, that you do not do this. You instead plow ahead:

```
. xtset roomtype 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 xtset will complain.

By the way, it is important only 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 p.m.), which is then followed by the usual 00:00:00 (12:00:00 a.m.). 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 for a set of patients, taken hourly at 1:00, 2:00, . . . , and that you have xtset 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 xtset 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 xtset that. If variable T contains the %tC value,

```
. generate double t = cofC(T)
. format t %tc
. xtset panelvar t, delta(...)
```

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

❑

Stored results

xtset 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
r(gaps)	1 if there are gaps, 0 otherwise

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

Also see

[XT] **xtdescribe** — Describe pattern of xt data

[XT] **xtsum** — Summarize xt data

[TS] **tsset** — Declare data to be time-series data

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