tsset — Declare data to be time-series data

Description

*tsset* manages the time-series settings of a dataset. *tsset* *timevar* declares the data in memory to be a time series. This allows you to use Stata’s time-series operators and to analyze your data with the *ts* commands. *tsset* *panelvar* *timevar* declares the data to be panel data, also known as cross-sectional time-series data, which contain one time series for each value of *panelvar*. This allows you to also analyze your data with the *xt* commands without having to *xtset* your data.

*tsset* without arguments displays how the data are currently set and sorts the data on *timevar* or *panelvar* timevar.

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

Quick start

Declare data to be a time series with time variable *tvar*

\[\text{tsset tvar}\]

As above, but specify that *tvar* records time for a weekly time series

\[\text{tsset tvar, weekly}\]

As above, but specify that observations occur every two weeks

\[\text{tsset tvar, weekly delta(2)}\]

Declare a panel dataset with panel identifier *pvar* and time variable *tvar*

\[\text{tsset pvar tvar}\]

As above, but specify that observations on each panel are made daily

\[\text{tsset pvar tvar, daily}\]

As above, but specify that observations on each panel are made every three days

\[\text{tsset pvar tvar, daily delta(3 days)}\]

Display current time-series settings, and sort data by *pvar* and *tvar* if they are sorted differently

\[\text{tsset}\]

Menu

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

Declare data to be time series

\[ \text{tsset } \text{timevar} \left[ , \text{options} \right] \]
\[ \text{tsset } \text{panelvar} \text{ timevar} \left[ , \text{options} \right] \]

Display how data are currently tsset

\[ \text{tsset} \]

Clear time-series settings

\[ \text{tsset, clear} \]

In the declare syntax, \textit{panelvar} identifies the panels and \textit{timevar} identifies the times.

\begin{tabular}{ll}
\textbf{options} & Description \\
\hline
Main & \\
\textit{unitoptions} & specify units of \textit{timevar} \\
Delta & \\
\textit{deltaoption} & specify length of period of \textit{timevar} \\
\textit{noquery} & suppress summary calculations and output \\
\end{tabular}

\textit{noquery} is not shown in the dialog box.

\begin{tabular}{ll}
\textit{unitoptions} & Description \\
\hline
\texttt{(default)} & \textit{timevar}'s units from \textit{timevar}'s display format \\
\texttt{clocktime} & \textit{timevar} is \texttt{1960 00:00:00}, \texttt{1961 00:00:00}, \ldots \\
\texttt{daily} & \textit{timevar} is \texttt{1960 01jan}, \texttt{1960 02jan}, \ldots \\
\texttt{weekly} & \textit{timevar} is \texttt{1960w1}, \texttt{1960w2}, \ldots \\
\texttt{monthly} & \textit{timevar} is \texttt{1960m1}, \texttt{1960m2}, \ldots \\
\texttt{quarterly} & \textit{timevar} is \texttt{1960q1}, \texttt{1960q2}, \ldots \\
\texttt{halfyearly} & \textit{timevar} is \texttt{1960h1}, \texttt{1960h2}, \ldots \\
\texttt{yearly} & \textit{timevar} is \texttt{1960 1961}, \texttt{1961 1962}, \ldots \\
\texttt{generic} & \textit{timevar} is \texttt{1960 1961}, \texttt{1961 1962}, \ldots \\
\texttt{format(\%fmt)} & \textit{timevar}'s format and then apply default rule \\
\end{tabular}

In all cases, negative \textit{timevar} values are allowed.
*deltaoption* specifies the period between observations in *timevar* units and may be specified as

<table>
<thead>
<tr>
<th><em>deltaoption</em></th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delta(#)</code></td>
<td>delta(1) or delta(2)</td>
</tr>
<tr>
<td><code>delta((exp))</code></td>
<td>delta((7*24))</td>
</tr>
<tr>
<td><code>delta(# units)</code></td>
<td>delta(7 days) or delta(15 minutes) or delta(7 days 15 minutes)</td>
</tr>
<tr>
<td><code>delta((exp) units)</code></td>
<td>delta((2+3) weeks)</td>
</tr>
</tbody>
</table>

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.

**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 `%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:

<table>
<thead>
<tr>
<th>Alternative 1</th>
<th>Alternative 2</th>
<th>Alternative 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>format t %td</code></td>
<td>(t not formatted)</td>
<td>(t not formatted)</td>
</tr>
<tr>
<td><code>tsset t</code></td>
<td><code>tsset t, daily</code></td>
<td><code>tsset t, format(%td)</code></td>
</tr>
</tbody>
</table>

*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()* 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.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 length of period. The `unitoptions` set the units, `delta()` sets the length of period.

We mentioned that `delta()` is commonly used with `\%tc` variables 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 `{..., -4, -2, 0, 2, 4, ...}` or it could contain any subset of `{..., -3, -1, 1, 3, ...}`. If `timevar` contained a mix of values, `tsset` would issue an error message. If you also specify `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

Remarks are presented under the following headings:

- **Overview**
- **Panel data**
- **Video example**
Overview

tset sets timevar so that Stata’s time-series operators are understood in varlists and expressions. The time-series operators are

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.</td>
<td>lag $x_{t-1}$</td>
</tr>
<tr>
<td>L2.</td>
<td>2-period lag $x_{t-2}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>F.</td>
<td>lead $x_{t+1}$</td>
</tr>
<tr>
<td>F2.</td>
<td>2-period lead $x_{t+2}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>D.</td>
<td>difference $x_t - x_{t-1}$</td>
</tr>
<tr>
<td>D2.</td>
<td>difference of difference $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>S.</td>
<td>“seasonal” difference $x_t - x_{t-1}$</td>
</tr>
<tr>
<td>S2.</td>
<td>lag-2 (seasonal) difference $x_t - x_{t-2}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

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. ≠ S2., D3. ≠ 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 ld21s12.d.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)
```

<table>
<thead>
<tr>
<th>year</th>
<th>L.gnp</th>
<th>L2.gnp</th>
<th>L3.gnp</th>
<th>L.cpi</th>
<th>L2.cpi</th>
<th>L3.cpi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1989</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1990</td>
<td>5452.8</td>
<td>.</td>
<td>.</td>
<td>100</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1991</td>
<td>5764.9</td>
<td>5452.8</td>
<td>.</td>
<td>105</td>
<td>100</td>
<td>.</td>
</tr>
<tr>
<td>1992</td>
<td>5932.4</td>
<td>5764.9</td>
<td>5452.8</td>
<td>108</td>
<td>105</td>
<td>100</td>
</tr>
<tr>
<td>1996</td>
<td>7330.1</td>
<td>6892.2</td>
<td>6519.1</td>
<td>122</td>
<td>119</td>
<td>112</td>
</tr>
</tbody>
</table>

In operator#, making # zero returns the variable itself. L0.gnp is gnp. Thus, you can type list year 1(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. Such datasets can be analyzed with xt commands as well as ts commands. If you tsset panelvar timevar, you do not need to xtset the data to use the xt commands.

If you save the data after typing tsset, the data will be remembered to be time series, and you will not have to tsset the data again.

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/r15/tssetxmpl
. list t income

<table>
<thead>
<tr>
<th>t</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1153</td>
</tr>
<tr>
<td>2</td>
<td>1181</td>
</tr>
<tr>
<td>9</td>
<td>1282</td>
</tr>
</tbody>
</table>

. tsset t
time variable: t, 1 to 9
delta: 1 unit
. regress income l.income

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 \(-3, -2, \ldots, 5, \) or \( 1,023, 1,024, \ldots, 1,031 \). What is important is that the difference in \( t \) between observations be \texttt{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 \texttt{[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

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

. list t newt income

<table>
<thead>
<tr>
<th>t</th>
<th>newt</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>426</td>
<td>1153</td>
</tr>
<tr>
<td>2</td>
<td>427</td>
<td>1181</td>
</tr>
<tr>
<td>3</td>
<td>428</td>
<td>1208</td>
</tr>
<tr>
<td>9</td>
<td>434</td>
<td>1282</td>
</tr>
</tbody>
</table>

If we put a \%tm format on \texttt{newt}, it will display more cleanly:

. format newt %tm
. list t newt income

<table>
<thead>
<tr>
<th>t</th>
<th>newt</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>2</td>
<td>1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>3</td>
<td>1995m9</td>
<td>1208</td>
</tr>
<tr>
<td>9</td>
<td>1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>
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
  
  ```
  . generate newt = td(15mar1993) + t - 1
  . tsset newt, daily
  ```

- **Weekly:** if your `t` variable had `t=1` corresponding to 1994w1:
  
  ```
  . generate newt = tw(1994w1) + t - 1
  . tsset newt, weekly
  ```

- **Monthly:** if your `t` variable had `t=1` corresponding to 2004m7:
  
  ```
  . generate newt = tm(2004m7) + t - 1
  . tsset newt, monthly
  ```

- **Quarterly:** if your `t` variable had `t=1` corresponding to 1994q1:
  
  ```
  . generate newt = tq(1994q1) + t - 1
  . tsset newt, quarterly
  ```

- **Half-yearly:** if your `t` variable had `t=1` corresponding to 1921h2:
  
  ```
  . generate newt = th(1921h2) + t - 1
  . tsset newt, halfyearly
  ```

- **Yearly:** if your `t` variable had `t=1` corresponding to 1842:
  
  ```
  . generate 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/r15/tssetxmpl2, clear
. list income

<table>
<thead>
<tr>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 1153</td>
</tr>
<tr>
<td>2. 1181</td>
</tr>
<tr>
<td>3. 1208</td>
</tr>
<tr>
<td>4. 1272</td>
</tr>
<tr>
<td>5. 1236</td>
</tr>
<tr>
<td>6. 1297</td>
</tr>
<tr>
<td>7. 1265</td>
</tr>
<tr>
<td>8. 1230</td>
</tr>
<tr>
<td>9. 1282</td>
</tr>
</tbody>
</table>

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 \texttt{tsset} our dataset and \texttt{list} it:

. tsset t
  \hspace{1cm} \text{time variable: t, 1995m7 to 1996m3}
  \hspace{1cm} \text{delta: 1 month}
. list t income

<table>
<thead>
<tr>
<th>t</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 1995m7</td>
<td>1153</td>
</tr>
<tr>
<td>2. 1995m8</td>
<td>1181</td>
</tr>
<tr>
<td>3. 1995m9</td>
<td>1208</td>
</tr>
<tr>
<td>4. 1996m3</td>
<td>1282</td>
</tr>
</tbody>
</table>
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/r15/tssetxmpl, clear
. list yrmo income
    +------------------+---------+
    |       yrmo    | income |     
    +------------------+---------+
    |  7/1995        |   1153  |
    |  8/1995        |   1181  |
    |  9-1995        |   1208  |
    | 10,1995        |   1272  |
    |   11,1995      |   1236  |
    |  12 1995       |   1297  |
    |  1/1996        |   1265  |
    |  2.1996        |   1230  |
    |  3-1 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`.

```
. generate mdate = monthly(yrmo, "MY")
. list yrmo mdate income
    +------------------+---------+---------+
    |       yrmo   |   mdate |    income |
    +------------------+---------+---------+
    |  7/1995        |      426 |   1153   |
    |  8/1995        |      427 |   1181   |
    |  9-1995        |      428 |   1208   |
    | 10,1995        |      429 |   1272   |
    |   11,1995      |      430 |   1236   |
    |  12 1995       |      431 |   1297   |
    |  1/1996        |      432 |   1265   |
    |  2.1996        |      433 |   1230   |
    |  3-1 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|
    |-------|-----------|-------|
    | 7/1995| 1995m7    | 1153  |
    | 8/1995| 1995m8    | 1181  |
    | 9-1995| 1995m9    | 1208  |
    | 10-1995| 1995m10  | 1272  |
    | 11 1995| 1995m11  | 1236  |
    | 12 1995| 1995m12  | 1297  |
    | 1/1996| 1996m1    | 1265  |
    | 2.1996| 1996m2    | 1230  |
    | 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/r15/tssetxmpl3, clear
. list yrmo income
    +-----------------+-------+
    | yrmo  | income|
    |-------|-------|
    | 7/1995| 1153  |
    | 8/1995| 1181  |
    | 11 1995| 1236 |
    | 12 1995| 1297 |
    | 1/1996| 1265  |
    | 3- 1996| 1282 |
    +-----------------+-------+
```

. generate 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 |
    |--------|-------|----------|
    | 1995m7 | 1153  | .        |
    | 1995m8 | 1181  | 28       |
    | 1995m11| 1236  | .        |
    | 1995m12| 1297  | 61       |
    | 1996m1 | 1265  | -32      |
    | 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/r15/tssetxmpl4, clear
. list in 1/5
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007</td>
<td>140</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007</td>
<td>140</td>
<td>1.487e+12</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>1.487e+12</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
<td>1.487e+12</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>1.487e+12</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
<td>1.487e+12</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007</td>
<td>140</td>
<td>1,486,972,800,000</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>1,486,976,400,000</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
<td>1,486,980,000,000</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>155</td>
<td>1,486,983,600,000</td>
</tr>
<tr>
<td>02.13.2007</td>
<td>160</td>
<td>1,486,987,200,000</td>
</tr>
</tbody>
</table>
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 length of the periods 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/r15/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
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007 08:00</td>
<td>140</td>
<td>1.49e+12</td>
</tr>
<tr>
<td>02.13.2007 09:00</td>
<td>155</td>
<td>1.49e+12</td>
</tr>
<tr>
<td>02.13.2007 10:00</td>
<td>160</td>
<td>1.49e+12</td>
</tr>
<tr>
<td>02.13.2007 11:00</td>
<td>155</td>
<td>1.49e+12</td>
</tr>
<tr>
<td>02.13.2007 12:00</td>
<td>160</td>
<td>1.49e+12</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007 08:00</td>
<td>140</td>
<td>1,486,972,780,544</td>
</tr>
<tr>
<td>02.13.2007 09:00</td>
<td>155</td>
<td>1,486,976,450,560</td>
</tr>
<tr>
<td>02.13.2007 10:00</td>
<td>160</td>
<td>1,486,979,989,504</td>
</tr>
<tr>
<td>02.13.2007 11:00</td>
<td>155</td>
<td>1,486,983,659,520</td>
</tr>
<tr>
<td>02.13.2007 12:00</td>
<td>160</td>
<td>1,486,987,198,464</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>time</th>
<th>price</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.13.2007 08:00</td>
<td>140</td>
<td>13feb2007 07:59:40</td>
</tr>
<tr>
<td>02.13.2007 09:00</td>
<td>155</td>
<td>13feb2007 09:00:50</td>
</tr>
<tr>
<td>02.13.2007 10:00</td>
<td>160</td>
<td>13feb2007 09:59:49</td>
</tr>
<tr>
<td>02.13.2007 11:00</td>
<td>155</td>
<td>13feb2007 11:00:59</td>
</tr>
<tr>
<td>02.13.2007 12:00</td>
<td>160</td>
<td>13feb2007 11:59:58</td>
</tr>
</tbody>
</table>

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:

```stata
.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 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 `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,

```stata
.generate 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/r15/tssetxmpl5, clear
. list edlevel year income, sep(0)
<table>
<thead>
<tr>
<th>edlevel</th>
<th>year</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1988</td>
<td>14500</td>
</tr>
<tr>
<td>1</td>
<td>1989</td>
<td>14750</td>
</tr>
<tr>
<td>1</td>
<td>1990</td>
<td>14950</td>
</tr>
<tr>
<td>1</td>
<td>1991</td>
<td>15100</td>
</tr>
<tr>
<td>2</td>
<td>1989</td>
<td>22100</td>
</tr>
<tr>
<td>2</td>
<td>1990</td>
<td>22200</td>
</tr>
<tr>
<td>2</td>
<td>1992</td>
<td>22800</td>
</tr>
</tbody>
</table>
```

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)
<table>
<thead>
<tr>
<th>edlevel</th>
<th>year</th>
<th>income</th>
<th>D.income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1988</td>
<td>14500</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>1989</td>
<td>14750</td>
<td>250</td>
</tr>
<tr>
<td>1</td>
<td>1990</td>
<td>14950</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>1991</td>
<td>15100</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>1989</td>
<td>22100</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>1990</td>
<td>22200</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>1992</td>
<td>22800</td>
<td>.</td>
</tr>
</tbody>
</table>
```

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

Formatting and managing dates

Stored results

tset 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

References


Also see

[TS] tssfill — Fill in gaps in time variable