Title

stset — Declare data to be survival-time data

Syntax

Single-record-per-subject survival data

```
stset timevar [if] [weight] [, single_options]

streset [if] [weight] [, single_options]

st [, nocmd notable]

stset, clear
```

Multiple-record-per-subject survival data

```
stset timevar [if] [weight] , id(idvar) failure(failvar[==numlist])
    [multiple_options]

streset [if] [weight] [, multiple_options]

streset, {past|future|past future}

st [, nocmd notable]

stset, clear
```

single_options	Description
Main	
`failure(`*failvar*[`==`*numlist*]`)`	failure event
`noshow`	prevent other st commands from showing st setting information
Options	
`origin(time` *exp*`)`	define when a subject becomes at risk
`scale(#)`	rescale time value
`enter(time` *exp*`)`	specify when subject first enters study
`exit(time` *exp*`)`	specify when subject exits study
Advanced	
`if(`*exp*`)`	select records for which *exp* is true; recommended rather than `if` *exp*
`time0(`*varname*`)`	mechanical aspect of interpretation about records in dataset; seldom used

multiple_options	Description
Main	
* `id(`*idvar*`)`	multiple-record ID variable
* `failure(`*failvar* [`==`*numlist*]`)`	failure event
`noshow`	prevent other st commands from showing st setting information
Options	
`origin(`[*varname*`==`*numlist*] `time` *exp* | `min)`	define when a subject becomes at risk
`scale(`*#*`)`	rescale time value
`enter(`[*varname*`==`*numlist*] `time` *exp*`)`	specify when subject first enters study
`exit(failure` | [*varname*`==`*numlist*] `time` *exp*`)`	specify when subject exits study
Advanced	
`if(`*exp*`)`	select records for which *exp* is true; recommended rather than `if` *exp*
`ever(`*exp*`)`	select subjects for which *exp* is ever true
`never(`*exp*`)`	select subjects for which *exp* is never true
`after(`*exp*`)`	select records within subject on or after the first time *exp* is true
`before(`*exp*`)`	select records within subject before the first time *exp* is true
`time0(`*varname*`)`	mechanical aspect of interpretation about records in dataset; seldom used

* `id()` and `failure()` are required with `stset` multiple-record-per-subject survival data.

`fweight`s, `iweight`s, and `pweight`s are allowed; see [U] 11.1.6 weight.

Examples

```
. stset ftime                                    (time measured from 0, all failed)
. stset ftime, failure(died)                     (time measured from 0, censoring)
. stset ftime, failure(died) id(id)              (time measured from 0, censoring & ID)
. stset ftime, failure(died==2,3)                (time measured from 0, failure codes)
. stset ftime, failure(died) origin(time dob)    (time measured from dob, censoring)
```

You cannot harm your data by using `stset`, so feel free to experiment.

Menu

Statistics > Survival analysis > Setup and utilities > Declare data to be survival-time data

Description

st refers to survival-time data, which are fully described below.

`stset` declares the data in memory to be st data, informing Stata of key variables and their roles in a survival-time analysis. When you `stset` your data, `stset` runs various data consistency checks to ensure that what you have declared makes sense. If the data are weighted, you specify the weights when you `stset` the data, not when you issue the individual st commands.

`streset` changes how the st dataset is declared. In multiple-record data, `streset` can also temporarily set the sample to include records from before the time at risk (called the past) and records after failure (called the future). Then typing `streset` without arguments resets the sample back to the analysis sample.

`st` displays how the dataset is currently declared.

Whenever you type `stset` or `streset`, Stata runs or reruns data consistency checks to ensure that what you are now declaring (or declared in the past) makes sense. Thus if you have made any changes to your data or simply wish to verify how things are, you can type `streset` with no options.

`stset, clear` is for use by programmers. It causes Stata to forget the st markers, making the data no longer st data to Stata. The data remain unchanged. It is not necessary to `stset, clear` before doing another `stset`.

Options for use with stset and streset

Main

`id(`*idvar*`)` specifies the subject-ID variable; observations with equal, nonmissing values of *idvar* are assumed to be the same subject. *idvar* may be string or numeric. Observations for which *idvar* is missing (`.` or `""`) are ignored.

When `id()` is not specified, each observation is assumed to represent a different subject and thus constitutes a one-record-per-subject survival dataset.

When you specify `id()`, the data are said to be multiple-record data, even if it turns out that there is only one record per subject. Perhaps they would better be called potentially multiple-record data.

If you specify `id()`, `stset` requires that you specify `failure()`.

Specifying `id()` never hurts; we recommend it because a few st commands, such as `stsplit`, require an ID variable to have been specified when the dataset was `stset`.

`failure(`*failvar*[`==`*numlist*]`)` specifies the failure event.

If `failure()` is not specified, all records are assumed to end in failure. This is allowed with single-record data only.

If `failure(`*failvar*`)` is specified, *failvar* is interpreted as an indicator variable; 0 and missing mean censored, and all other values are interpreted as representing failure.

If `failure(`*failvar*`==`*numlist*`)` is specified, records with *failvar* taking on any of the values in *numlist* are assumed to end in failure, and all other records are assumed to be censored.

`noshow` prevents other st commands from showing the key st variables at the top of their output.

Options

`origin(`[*varname*`==`*numlist*] `time` *exp* | `min)` and `scale(`*#*`)` define analysis time; that is, `origin()` defines when a subject becomes at risk. Subjects become at risk when time = `origin()`. All analyses are performed in terms of time since becoming at risk, called analysis time.

Let us use the terms *time* for how time is recorded in the data and t for analysis time. Analysis time t is defined

$$t = \frac{\textit{time} - \texttt{origin()}}{\texttt{scale()}}$$

t is time from origin in units of scale.

By default, `origin(time 0)` and `scale(1)` are assumed, meaning that $t = time$. Then you must ensure that *time* in your data is measured as time since becoming at risk. Subjects are exposed at $t = time = 0$ and later fail. Observations with $t = time \leq 0$ are ignored because information before becoming at risk is irrelevant.

`origin()` determines when the clock starts ticking. `scale()` plays no substantive role, but it can be handy for making t units more readable (such as converting days to years).

`origin(time` *exp*`)` sets the origin to *exp*. For instance, if *time* were recorded as dates, such as 05jun1998, in your data and variable `expdate` recorded the date when subjects were exposed, you could specify `origin(time expdate)`. If instead all subjects were exposed on 12nov1997, you could specify `origin(time mdy(11,12,1997))`.

`origin(time` *exp*`)` may be used with single- or multiple-record data.

`origin(`*varname*`==`*numlist*`)` is for use with multiple-record data; it specifies the origin indirectly. If *time* were recorded as dates in your data, variable `obsdate` recorded the (ending) date associated with each record, and subjects became at risk upon, say, having a certain operation—and that operation were indicated by `code==217`—then you could specify `origin(code==217)`. `origin(code==217)` would mean, for each subject, that the origin time is the earliest time at which `code==217` is observed. Records before that would be ignored (because $t < 0$). Subjects who never had `code==217` would be ignored entirely.

`origin(`*varname*`==`*numlist* `time` *exp*`)` sets the origin to the later of the two times determined by *varname*`==`*numlist* and *exp*.

`origin(min)` sets origin to the earliest time observed, minus 1. This is an odd thing to do and is described in example 10.

`origin()` is an important concept; see *Key concepts*, *Two concepts of time*, and *The substantive meaning of analysis time*.

`scale()` makes results more readable. If you have *time* recorded in days (such as Stata dates, which are really days since 01jan1960), specifying `scale(365.25)` will cause results to be reported in years.

`enter(`[*varname*`==`*numlist*] `time` *exp*`)` specifies when a subject first comes under observation, meaning that any failures, were they to occur, would be recorded in the data.

Do not confuse `enter()` and `origin()`. `origin()` specifies when a subject first becomes at risk. In many datasets, becoming at risk and coming under observation are coincident. Then it is sufficient to specify `origin()`.

`enter(time` *exp*`)`, `enter(`*varname*`==`*numlist*`)`, and `enter(`*varname*`==`*numlist* `time` *exp*`)` follow the same syntax as `origin()`. In multiple-record data, both *varname*`==`*numlist* and `time` *exp* are interpreted as the earliest time implied, and if both are specified, the later of the two times is used.

`exit(failure` | [*varname*`==`*numlist*] `time` *exp*`)` specifies the latest time under which the subject is both under observation and at risk. The emphasis is on latest; obviously, subjects also exit the risk pool when their data run out.

`exit(failure)` is the default. When the first failure event occurs, the subject is removed from the analysis risk pool, even if the subject has subsequent records in the data and even if some of those subsequent records document other failure events. Specify `exit(time .)` if you wish to keep all records for a subject after failure. You want to do this if you have multiple-failure data.

`exit(`*varname*`==`*numlist*`)`, `exit(time` *exp*`)`, and `exit(`*varname*`==`*numlist* `time` *exp*`)` follow the same syntax as `origin()` and `enter()`. In multiple-record data, both *varname*`==`*numlist* and time *exp* are interpreted as the earliest time implied. exit differs from origin() and enter() in that if both are specified, the earlier of the two times is used.

Advanced

if(*exp*), ever(*exp*), never(*exp*), after(*exp*), and before(*exp*) select relevant records.

if(*exp*) selects records for which *exp* is true. We strongly recommend specifying this if() option rather than if *exp* following stset or streset. They differ in that if *exp* removes the data from consideration before calculating beginning and ending times and other quantities. The if() option, on the other hand, sets the restriction after all derived variables are calculated. See *if() versus if exp*.

if() may be specified with single- or multiple-record data. The remaining selection options are for use with multiple-record data only.

ever(*exp*) selects only subjects for which *exp* is ever true.

never(*exp*) selects only subjects for which *exp* is never true.

after(*exp*) selects records within subject on or after the first time *exp* is true.

before(*exp*) selects records within subject before the first time *exp* is true.

time0(*varname*) is seldom specified because most datasets do not contain this information. time0() should be used exclusively with multiple-record data, and even then you should consider whether origin() or enter() would be more appropriate.

time0() specifies a mechanical aspect of interpretation about the records in the dataset, namely, the beginning of the period spanned by each record. See *Intermediate exit and reentry times (gaps)*.

Options unique to streset

past expands the stset sample to include the entire recorded past of the relevant subjects, meaning that it includes observations before becoming at risk and those excluded because of after(), etc.

future expands the stset sample to include the records on the relevant subjects after the last record that previously was included, if any, which typically means to include all observations after failure or censoring.

past future expands the stset sample to include all records on the relevant subjects.

Typing streset without arguments resets the sample to the analysis sample. See *Past and future records* for more information.

Options for st

nocmd suppresses displaying the last stset command.

notable suppresses displaying the table summarizing what has been stset.

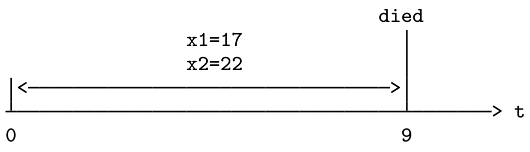
Remarks and examples

stata.com

Remarks are presented under the following headings:

What are survival-time data?

Survival-time data—what we call st data—document spans of time ending in an event. For instance,

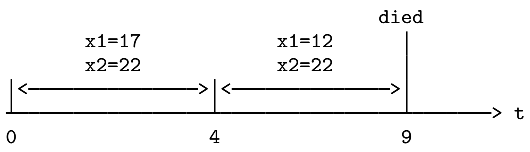


which indicates `x1` = 17 and `x2` = 22 over the time span 0 to 9, and `died` = 1. More formally, it means `x1` = 17 and `x2` = 22 for $0 < t \leq 9$, which we often write as $(0, 9\,]$. However you wish to say it, this information might be recorded by the observation

id	end	x1	x2	died
101	9	17	22	1

and we call this single-record survival data.

The data can be more complicated. For instance, we might have



meaning

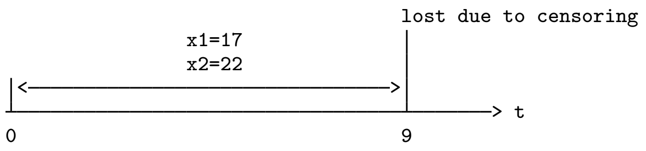
`x1` = 17 and `x2` = 22 during $(0, 4\,]$
`x1` = 12 and `x2` = 22 during $(4, 9\,]$, and then `died` = 1.

and this would be recorded by the data

id	begin	end	x1	x2	died
101	0	4	17	22	0
101	4	9	12	22	1

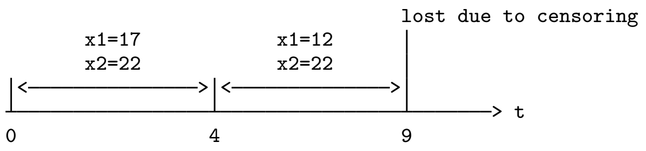
We call this multiple-record survival data.

These two formats allow you to record many different possibilities. The last observation on a person need not be failure,



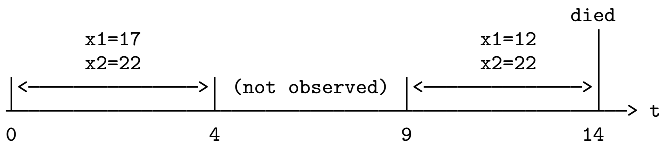
id	end	x1	x2	died
101	9	17	22	0

or



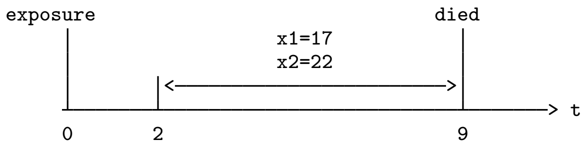
id	begin	end	x1	x2	died
101	0	4	17	22	0
101	4	9	12	22	0

Multiple-record data might have gaps,



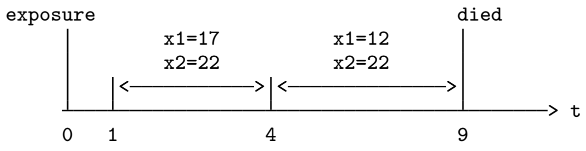
id	begin	end	x1	x2	died
101	0	4	17	22	0
101	9	14	12	22	1

or subjects might not be observed from the onset of risk,



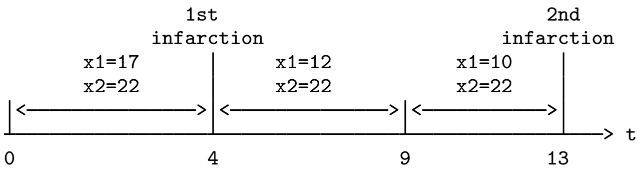
begin	end	x1	x2	died
2	9	17	22	1

and



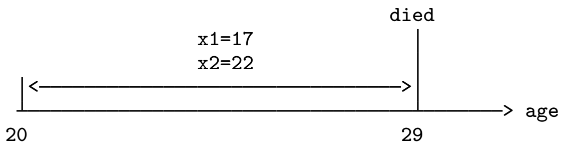
id	begin	end	x1	x2	died
101	1	4	17	22	0
101	4	9	12	22	1

The failure event might not be death but instead something that can be repeated:



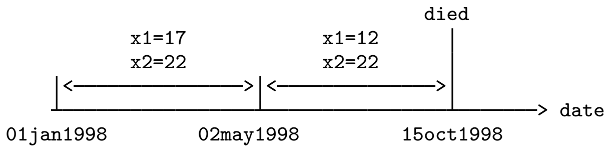
id	begin	end	x1	x2	infarc
101	0	4	17	22	1
101	4	9	12	22	0
101	9	13	10	22	1

Our data may be in different time units; rather than t where $t = 0$ corresponds to the onset of risk, we might have time recorded as age,



id	age0	age1	x1	x2	died
101	20	29	17	22	1

or time recorded as calendar dates:

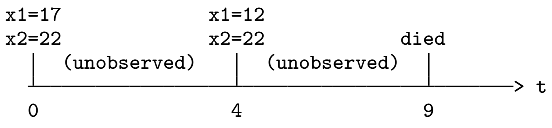


id	bdate	edate	x1	x2	died
101	01jan1998	02may1998	17	22	0
101	02may1998	15oct1998	12	22	1

Finally, we can mix these diagrams however we wish, so we might have time recorded according to the calendar, unobserved periods after the onset of risk, subsequent gaps, and multiple failure events.

The st commands analyze data like these, and the first step is to tell st about your data by using `stset`. You do not change your data to fit some predefined mold; you describe your data with `stset`, and the rest of the st commands just do the right thing.

Before we discuss using `stset`, let's describe one more style of recording time-to-event data because it is common and is inappropriate for use with st. It is inappropriate, but it is easy to convert to the survival-time form. It is called snapshot data, which are data for which you do not know spans of time but you have information recorded at various points in time:



id	t	x1	x2	died
101	0	17	22	0
101	4	12	22	0
101	9	.	.	1

In this snapshot dataset all we know are the values of `x1` and `x2` at $t = 0$ and $t = 4$, and we know that the subject died at $t = 9$. Snapshot data can be converted to survival-time data if we are willing to assume that `x1` and `x2` remained constant between times:

```
                                                   died
                   x1=17               x1=12        |
                   x2=22               x2=22        |
              |<------------------>|<------------------>|
              |                    |                    |
              ------------------------------------------------->
              0                    4                    9

        id    begin    end    x1    x2    died
       101        0      4    17    22       0
       101        4      9    12    22       1
```

The `snapspan` command makes this conversion. If you have snapshot data, first see [ST] **snapspan** to convert it to survival-time data and then use `stset` to tell st about the converted data; see example 10 first.

Key concepts

time, or, better, *time units*, is how time is recorded in your data. It might be numbers (such as 0, 1, 2, ..., with time = 0 corresponding to some exposure event), a subject's age, or calendar time.

events are things that happen at an instant in time, such as being exposed to an environmental hazard, being diagnosed as myopic, becoming employed, being promoted, becoming unemployed, having a heart attack, and dying.

failure event is the event indicating failure as it is defined for analysis. This can be a single or compound event. The failure event might be when variable `dead` is 1, or it might be when variable `diag` is any of 115, 121, or 133.

at risk means the subject is at risk of the failure event occurring. For instance, if the failure event is becoming unemployed, a person must be employed. The subject is not at risk before being employed. Once employed, the subject becomes at risk; once again, the subject is no longer at risk once the failure event occurs. If subjects become at risk upon the occurrence of some event, it is called the exposure event. Gaining employment is the exposure event in our example.

origin is the time when the subject became at risk. If time is recorded as numbers such as 0, 1, 2, ..., with time = 0 corresponding to the exposure event, then origin = 0. Alternatively, origin might be the age of the subject when diagnosed or the date when the subject was exposed. Regardless, origin is expressed in time units.

scale is just a fixed number, typically 1, used in mapping time to analysis time t.

t, or *analysis time*, is (time − origin)/scale, meaning the time since onset of being at risk measured in scale units.

$t = 0$ corresponds to the onset of risk, and scale just provides a way to make the units of t more readable. You might have time recorded in days from 01jan1960 and want t recorded in years, in which case scale would be 365.25.

Time is how time is recorded in your data, and t is how time is reported in the analysis.

under observation means that, should the failure event occur, it would be observed and recorded in the data. Sometimes subjects are under observation only after they are at risk. This would be the case, for instance, if subjects enrolled in a study after being diagnosed with cancer and if, to enroll in the study, subjects were required to be diagnosed with cancer.

Being under observation does not mean that the subject is necessarily at risk. A subject may come under observation before being at risk, and in fact, a subject under observation may never become at risk.

entry time and *exit time* mark when a subject is first and last under observation. The emphasis here is on the words *first* and *last*; entry time and exit time do not record observational gaps; there is only one entry time and one exit time per subject.

Entry time and exit time might be expressed as times (recorded in time units), or they might correspond to the occurrence of some event (such as enrolling in the study).

Often the entry time corresponds to $t = 0$; that is, because $t = (\text{time} - \text{origin})/\text{scale}$, time = origin, meaning that time equals when the subject became at risk.

Often the exit time corresponds to when the failure event occurs or, failing that, the end of data for the subject.

delayed entry means that entry time corresponds to $t > 0$; the subject became at risk before coming under observation.

ID refers to a subject identification variable; equal values of ID indicate that the records are on the same subject. An ID variable is required for multiple-record data and is optional, but recommended, with single-record data.

time0 refers to the beginning time (recorded in time units) of a record. Some datasets have this variable, but most do not. If the dataset does not contain the beginning time for each record, subsequent records are assumed to begin where previous records ended. A time0 variable may be created for these datasets by using the `snapspan` command; see [ST] **snapspan**. Do not confuse time0—a mechanical aspect of datasets—with entry time—a substantive aspect of analysis.

gaps refer to gaps in observation between entry time and exit time. During a gap, a subject is not under observation. Gaps can arise only if the data contain a time0 variable, because otherwise subsequent records beginning when previous records end would preclude there being gaps in the data. Gaps are distinct from delayed entry.

past history is a term we use to mean information recorded in the data before the subject was both at risk and under observation. Complex datasets can contain such observations. Say that the dataset contains histories on subjects from birth to death. You might tell st that a subject becomes at risk once diagnosed with a particular kind of cancer. The past history on the subject would then refer to records before the subject was diagnosed.

The word *history* is often dropped, and the term simply becomes *past*. For instance, we might want to know whether the subject smoked in the past.

future history is a term meaning information recorded in the data after the subject is no longer at risk. Perhaps the failure event is not so serious as to preclude the possibility of data after failure.

The word *history* is often dropped, and the term simply becomes *future*. Perhaps the failure event is cardiac infarction, and you want to know whether the subject died soon in the future so that you can exclude the subject.

Survival-time datasets

In survival-time datasets, observations (records) document a span of time. The span might be explicitly indicated, such as

```
                begin    end     x1     x2
                    3      9     17     22              <- spans (3,9]
```

or it might be implied that the record begins at 0,

```
                          end     x1     x2
                            9     17     22            <- spans (0,9]
```

or it might be implied because there are multiple records per subject:

```
                   id     end     x1     x2
                    1       4     17     22            <- spans (0,4]
                    1       9     12     22            <- spans (4,9]
```

Records may have an event indicator:

```
               begin     end     x1     x2    died
                   3       9     17     22       1     <- spans (3,9], died at t=9

                         end     x1     x2    died
                           9     17     22       1     <- spans (0,9], died at t=9


                  id     end     x1     x2    died
                   1       4     17     22       0     <- spans (0,4],
                   1       9     12     22       1     <- spans (4,9], died at t=9
```

The first two examples are called single-record survival-time data because there is one record per subject.

The final example is called multiple-record survival-time data. There are two records for the subject with `id` = 1.

Either way, survival-time data document time spans. Characteristics are assumed to remain constant over the span, and the event is assumed to occur at the end of the span.

Using stset

Once you have `stset` your data, you can use the other st commands.

If you `save` your data after `stset`ting, you will not have to re-`stset` in the future; Stata will remember.

`stset` declares your data to be survival-time data. It does not change the data, although it does add a few variables to your dataset.

This means that you can re-`stset` your data as often as you wish. In fact, the `streset` command encourages this. Using complicated datasets often requires typing long `stset` commands, such as

```
. stset date, fail(event==27 28) origin(event==15) enter(event==22)
```

Later, you might want to try `fail(event==27)`. You could retype the `stset` command, making the substitution, or you could type

```
. streset, fail(event==27)
```

`streset` takes what you type, merges it with what you have previously declared with `stset`, and performs the combined `stset` command.

▷ Example 1: Single-record data

Generators are run until they fail. Here is some of our dataset:

```
. use http://www.stata-press.com/data/r13/kva
(Generator experiment)

. list in 1/3

     +-----------------------------+
     | failtime   load   bearings |
     |-----------------------------|
  1. |      100     15          0 |
  2. |      140     15          1 |
  3. |       97     20          0 |
     +-----------------------------+
```

The `stset` command for this dataset is

```
. stset failtime

     failure event:  (assumed to fail at time=failtime)
obs. time interval:  (0, failtime]
 exit on or before:  failure

------------------------------------------------------------------------------
       12  total observations
        0  exclusions
------------------------------------------------------------------------------
       12  observations remaining, representing
       12  failures in single-record/single-failure data
      896  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =       140
```

When you type `stset` *timevar*, *timevar* is assumed to be the time of failure. More generally, you will learn, *timevar* is the time of failure or censoring. Here *timevar* is `failtime`.

◁

▷ Example 2: Single-record data with censoring

Generators are run until they fail, but during the experiment, the room flooded, so some generators were run only until the flood. Here are some of our data:

```
. use http://www.stata-press.com/data/r13/kva2
(Generator experiment)

. list in 1/4

     +--------------------------------------+
     | failtime   load   bearings   failed |
     |--------------------------------------|
  1. |      100     15          0        1 |
  2. |      140     15          1        0 |
  3. |       97     20          0        1 |
  4. |      122     20          1        1 |
     +--------------------------------------+
```

Here the second generator did not fail at time 140; the experiment was merely discontinued then. The `stset` command for this dataset is

```
. stset failtime, failure(failed)

     failure event:  failed != 0 & failed < .
obs. time interval:  (0, failtime]
 exit on or before:  failure

──────────────────────────────────────────────────────────────────────────────
         12  total observations
          0  exclusions
──────────────────────────────────────────────────────────────────────────────
         12  observations remaining, representing
         11  failures in single-record/single-failure data
        896  total analysis time at risk and under observation
                                                  at risk from t =         0
                                       earliest observed entry t =         0
                                            last observed exit t =       140
```

When you type `stset` *timevar*`, failure(`*failvar*`)`, *timevar* is interpreted as the time of failure or censoring, which is determined by the value of *failvar*. *failvar* $= 0$ and *failvar* $= .$ (missing) indicate censorings, and all other values indicate failure.

◁

▷ Example 3: Multiple-record data

Assume that we are analyzing survival time of patients with a particular kind of cancer. In this dataset, the characteristics of patients vary over time, perhaps because new readings were taken or because the drug therapy was changed. Some of the data are

```
. list, separator(0)

     ┌──────────────────────────────┐
     │ patid     t   died   x1   x2 │
     ├──────────────────────────────┤
  1. │    90   100      0    1    0 │
  2. │    90   150      1    0    0 │
  3. │    91    50      1    1    1 │
  4. │    92   100      0    0    0 │
  5. │    92   150      0    0    1 │
  6. │    92   190      0    0    0 │
  7. │    93   100      0    0    0 │
              (output omitted)
     └──────────────────────────────┘
```

There are two records for patient 90, and `died` is 0 in the first record but 1 in the second. The interpretation of these two records is

Interval (0,100]:	`x1` = 1 and `x2` = 0
Interval (100,150]:	`x1` = 0 and `x2` = 0
at `t` = 150:	the patient died

Similarly, here is how you interpret the other records:

Patient 91:

Interval (0, 50]:	`x1` = 1 and `x2` = 1
at `t` = 50:	the patient died

Patient 92:

Interval (0,100]:	`x1` = 0 and `x2` = 0
Interval (100,150]:	`x1` = 0 and `x2` = 1
Interval (150,190]:	`x1` = 0 and `x2` = 0
at `t` = 190:	the patient was lost because of censoring

Look again at patient 92's data:

```
                patid          t        died          x1          x2
                   92        100           0           0           0
                   92        150           0           0           1
                   92        190           0           0           0
```

`died` = 0 for the first event. Mechanically, this removes the subject from the data at `t` = 100—the patient is treated as censored. The next record, however, adds the patient back into the data (at `t` = 100) with new characteristics.

The `stset` command for this dataset is

```
. stset t, id(patid) failure(died)

                id:  patid
     failure event:  died != 0 & died < .
obs. time interval:  (t[_n-1], t]
 exit on or before:  failure

------------------------------------------------------------------------------
      126  total observations
        0  exclusions
------------------------------------------------------------------------------
      126  observations remaining, representing
       40  subjects
       26  failures in single-failure-per-subject data
     2989  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =       139
```

When you have multiple-record data, you specify `stset`'s `id(`*idvar*`)` option. When you type `stset` *timevar*`, id(`*idvar*`) failure(`*failvar*`)`, *timevar* denotes the end of the period (just as it does in single-record data). The first record within *idvar* is assumed to begin at time 0, and later records are assumed to begin where the previous record left off. *failvar* should contain 0 on all but, possibly, the last record within *idvar*, unless your data contain multiple failures (in which case you must specify the `exit()` option; see *Setting multiple failures* below).

◁

▷ Example 4: Multiple-record data with multiple events

We have the following data on hospital patients admitted to a particular ward:

```
      patid     day     sex      x1      x2    code
        101       5       1      10      10     177
        101      13       1      20       8     286
        101      21       1      16      11     208
        101      24       1      11      17     401
        102       8       0      20      19     204
        102      18       0      19       1     401
        103     etc.
```

Variable `code` records various actions; code 401 indicates being discharged alive, and 402 indicates death. We `stset` this dataset by typing

```
. stset day, id(patid) fail(code==402)

                id:  patid
     failure event:  code == 402
obs. time interval:  (day[_n-1], day]
 exit on or before:  failure

------------------------------------------------------------------------------
        243  total observations
          0  exclusions
------------------------------------------------------------------------------
        243  observations remaining, representing
         40  subjects
         15  failures in single-failure-per-subject data
       1486  total analysis time at risk and under observation
                                                 at risk from t =         0
                                      earliest observed entry t =         0
                                           last observed exit t =        62
```

When you specify `failure(`*eventvar*`==`*#*`)`, the failure event is as specified. You may include a list of numbers following the equal signs. If failure were codes 402 and 403, you could specify `failure(code == 402 403)`. If failure were codes 402, 403, 404, 405, 406, 407, and 409, you could specify `failure(code == 402/407 409)`.

◁

▷ Example 5: Multiple-record data recording time rather than t

More reasonably, the hospital data in the above example would not contain days since admission but would contain admission and current dates. In the dataset below, `adday` contains the day of admission, and `curday` contains the ending date of the record, both recorded as number of days since the ward opened:

```
        patid       adday      curday     sex      x1      x2    code
          101         287         292       1      10      10     177
          101           .         300       1      20       8     286
          101           .         308       1      16      11     208
          101           .         311       1      11      17     401
          102         289         297       0      20      19     204
          102           .         307       0      19       1     401
          103        etc.
```

This is the same dataset as shown in example 4. Previously, the first record on patient 101 was recorded 5 days after admission. In this dataset, 292 − 287 = 5. We would `stset` this dataset by typing

```
. stset curday, id(patid) fail(code==402) origin(time adday)
                 id:  patid
      failure event:  code == 402
obs. time interval:  (curday[_n-1], curday]
 exit on or before:  failure
    t for analysis:  (time-origin)
            origin:  time adday
──────────────────────────────────────────────────────────────────────────────
        243  total observations
          0  exclusions
──────────────────────────────────────────────────────────────────────────────
        243  observations remaining, representing
         40  subjects
         15  failures in single-failure-per-subject data
       1486  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =        62
```

`origin()` sets when a subject becomes at risk. It does this by defining analysis time.

When you specify `stset` *timevar*`, ... origin(time` *originvar*`)`, analysis time is defined as $t = (timevar - originvar)/$`scale()`. In analysis-time units, subjects become at risk at $t = 0$. See *Two concepts of time* and *The substantive meaning of analysis time* below.

◁

▷ Example 6: Multiple-record data with time recorded as a date

Even more reasonably, dates would not be recorded as integers 428, 433, and 453, meaning the number of days since the ward opened. The dates would be recorded as dates:

patid	addate	curdate	sex	x1	x2	code
101	18aug1998	23aug1998	1	10	10	177
101	.	31aug1998	1	20	8	286
101	.	08sep1998	1	16	11	208
101	.	11sep1998	1	11	17	401
102	20aug1998	28aug1998	0	20	19	204
102	.	07sep1998	0	19	1	401
103	*etc.*					

That, in fact, changes nothing. We still type what we previously typed:

```
. stset curdate, id(patid) fail(code==402) origin(time addate)
```

Stata dates are, in fact, integers—they are the number of days since 01jan1960—and it is merely Stata's `%td` display format that makes them display as dates.

◁

▷ Example 7: Multiple-record data with extraneous information

Perhaps we wish to study the outcome after a certain operation, said operation being indicated by code 286. Subjects become at risk when the operation is performed. Here we do not type

```
. stset curdate, id(patid) fail(code==402) origin(time addate)
```

We instead type

```
. stset curdate, id(patid) fail(code==402) origin(code==286)
```

The result of typing this would be to set analysis time t to

$$t = \texttt{curdate} - (\text{the value of } \texttt{curdate} \text{ when } \texttt{code==286})$$

Let's work through this for the first patient:

```
    patid     addate     curdate     sex     x1     x2    code
      101  18aug1998   23aug1998       1     10     10     177
      101          .   31aug1998       1     20      8     286
      101          .   08sep1998       1     16     11     208
      101          .   11sep1998       1     11     17     401
```

The event 286 occurred on 31aug1998, and thus the values of t for the four records are

$$\begin{aligned} t_1 &= \mathit{curdate}_1 - 31\text{aug}1998 = 23\text{aug}1998 - 31\text{aug}1998 = -8 \\ t_2 &= \mathit{curdate}_2 - 31\text{aug}1998 = 31\text{aug}1998 - 31\text{aug}1998 = 0 \\ t_3 &= \mathit{curdate}_3 - 31\text{aug}1998 = 08\text{sep}1998 - 31\text{aug}1998 = 8 \\ t_4 &= \mathit{curdate}_4 - 31\text{aug}1998 = 11\text{sep}1998 - 31\text{aug}1998 = 11 \end{aligned}$$

Information prior to $t = 0$ is not relevant because the subject is not yet at risk. Thus the relevant data on this subject are

t in $(0, 8\,]$	sex = 1, x1 = 16, x2 = 11
t in $(8, 11\,]$	sex = 1, x1 = 11, x2 = 17, and the subject is censored (code $\neq$ 402)

That is precisely the logic that stset went through. For your information, stset quietly creates the variables

_st	1 if the record is to be used, 0 if ignored
_t0	analysis time when record begins
_t	analysis time when record ends
_d	1 if failure, 0 if censored

You can examine these variables after issuing the stset command:

```
. list _st _t0 _t _d

        +-----------------------+
        | _st   _t0   _t    _d  |
        |-----------------------|
     1. |   0     .    .     .  |
     2. |   0     .    .     .  |
   203. |   1     0    8     0  |
   204. |   1     8   11     0  |
        +-----------------------+
```

Results are just as we anticipated. Do not let the observation numbers bother you; stset sorts the data in a way it finds convenient. Feel free to re-sort the data; if any of the st commands need the data in a different order, they will sort it themselves.

There are two ways of specifying origin():

origin(time *timevar*)   or   origin(time *exp*)
origin(*eventvar* == *numlist*)

In the first syntax—which is denoted by typing the word time—you directly specify when a subject becomes at risk. In the second syntax—which is denoted by typing a variable name and equal signs—you specify the same thing indirectly. The subject becomes at risk when the specified event occurs (which may be never).

Information prior to `origin()` is ignored. That information composes what we call the past history.

◁

▷ Example 8: Multiple-record data with delayed entry

In another analysis, we want to use the above data to analyze all patients, not just those undergoing a particular operation. In this analysis, subjects become at risk when they enter the ward. For this analysis, however, we need information from a particular test, and that information is available only if the test is administered to the patient. Even if the test is administered, some amount of time passes before that. Assume that when the test is administered, `code==152` is inserted into the patient's hospital record.

To summarize, we want `origin(time addate)`, but patients do not enter our sample until `code==152`. The way to `stset` these data is

```
. stset curdate, id(patid) fail(code==402) origin(time addate) enter(code==152)
```

Patient 107 has code 152:

```
patid     addate     curdate    sex    x1    x2   code
  107  22aug1998   25aug1998      1     9    13    274
  107          .   28aug1998      1    19    19    152
  107          .   30aug1998      1    18    12    239
  107          .   07sep1998      1    12    11    401
```

In analysis time, $t = 0$ corresponds to 22aug1998. The test was not administered, however, until 6 days later. The analysis times for these records are

$$t_1 = \mathit{curdate}_1 - \text{22aug1998} = \text{25aug1998} - \text{22aug1998} = 3$$
$$t_2 = \mathit{curdate}_2 - \text{22aug1998} = \text{28aug1998} - \text{22aug1998} = 6$$
$$t_3 = \mathit{curdate}_3 - \text{22aug1998} = \text{30aug1998} - \text{22aug1998} = 8$$
$$t_4 = \mathit{curdate}_4 - \text{22aug1998} = \text{07sep1998} - \text{22aug1998} = 16$$

and the data we want in our sample are

t in $(6, 8]$ `sex` $= 1$, `x1` $= 18$, `x2` $= 12$
t in $(8, 16]$ `sex` $= 1$, `x1` $= 12$, `x2` $= 11$, and patient was censored (`code` $\neq 402$)

The above `stset` command produced this:

```
. list _st _t0 _t _d

     +-----------------------+
     | _st   _t0    _t   _d  |
     |-----------------------|
  1. |   0     .     .    .  |
  2. |   0     .     .    .  |
 39. |   1     6     8    0  |
 40. |   1     8    16    0  |
     +-----------------------+
```

◁

Example 9: Multiple-record data with extraneous information and delayed entry

The `origin()` and `enter()` options can be combined. For instance, we want to analyze patients receiving a particular operation (time at risk begins upon `code` == 286, but patients may not enter the sample before a test is administered, denoted by `code` == 152). We type

```
. stset curdate, id(patid) fail(code==402) origin(code==286) enter(code==152)
```

If we typed the above commands, it would not matter whether the test was performed before or after the operation.

A patient who had the test and then the operation would enter at analysis time $t = 0$.

A patient who had the operation and then the test would enter at analysis time $t > 0$, the analysis time being the time the test was performed.

If we wanted to require that the operation be performed after the test, we could type

```
. stset curdate, id(patid) fail(code==402) origin(code==286) after(code==152)
```

Admittedly, this can be confusing. The way to proceed is to find a complicated case in your data and then list `_st _t0 _t _d` for that case after you `stset` the data.

Example 10: Real data

All of our hospital ward examples are artificial in one sense: it is unlikely the data would have come to us in survival-time form:

patid	addate	curdate	sex	x1	x2	code
101	18aug1998	23aug1998	1	10	10	177
101	.	31aug1998	1	20	8	286
101	.	08sep1998	1	16	11	208
101	.	11sep1998	1	11	17	401
102	20aug1998	28aug1998	0	20	19	204
102	.	07sep1998	0	19	1	401
103	*etc.*					

Rather, we would have received a snapshot dataset:

patid	date	sex	x1	x2	code
101	18aug1998	1	10	10	22
101	23aug1998	.	20	8	177
101	31aug1998	.	16	11	286
101	08sep1998	.	11	17	208
101	11sep1998	.	.	.	401
102	20aug1998	0	20	19	22
102	28aug1998	.	19	1	204
102	07sep1998	.	.	.	401
103	*etc.*				

In a snapshot dataset, we have a time (here a date) and values of the variables as of that instant.

This dataset can be converted to the appropriate form by typing

```
. snapspan patid date code
```

The result would be as follows:

```
        patid       date    sex      x1      x2    code
          101  18aug1998      .       .       .      22
          101  23aug1998      1      10      10     177
          101  31aug1998      .      20       8     286
          101  08sep1998      .      16      11     208
          101  11sep1998      .      11      17     401
          102  20aug1998      .       .       .      22
          102  28aug1998      0      20      19     204
          102  07sep1998      .      19       1     401
```

This is virtually the same dataset with which we have been working, but it differs in two ways:

1. The variable `sex` is not filled in for all the observations because it was not filled in on the original form. The hospital wrote down the sex on admission and then never bothered to document it again.
2. We have no admission date (`addate`) variable. Instead, we have an extra first record for each patient with `code` = 22 (22 is the code the hospital uses for admissions).

The first problem is easily fixed, and the second, it turns out, is not a problem because we can vary what we type when we `stset` the data.

First, let's fix the problem with variable `sex`. There are two ways to proceed. One would be simply to fill in the variable ourselves:

```
. by patid (date), sort: replace sex = sex[_n-1] if sex>=.
```

We could also perform a phony `stset` that is good enough to set all the data and then use `stfill` to fill in the variable for us. Let's begin with the phony `stset`:

```
. stset date, id(patid) origin(min) fail(code==-1)

                id:  patid
     failure event:  code == -1
obs. time interval:  (date[_n-1], date]
 exit on or before:  failure
    t for analysis:  (time-origin)
            origin:  min

------------------------------------------------------------------------------
      283  total observations
        0  exclusions
------------------------------------------------------------------------------
      283  observations remaining, representing
       40  subjects
        0  failures in single-failure-per-subject data
     2224  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =        89
```

Typing `stset date, id(patid) origin(min) fail(code==-1)` does not produce anything we would want to use for analysis. This is a trick to get the dataset temporarily `stset` so that we can use some st data management commands on it.

The first part of the trick is to specify `origin(min)`. This defines the analysis time as $t = 0$, corresponding to the minimum observed value of the time variable minus 1. The time variable is `date` here. Why the minimum minus 1? Because st ignores observations for which analysis time $t < 0$. `origin(min)` provides a phony definition of t that ensures $t > 0$ for all observations.

The second part of the trick is to specify `fail(code == -1)`, and you might have to vary what you type. We just wanted to choose an event that we know never happens, thus ensuring that no observations are ignored after failure.

Now that we have the dataset `stset`, we can use the other st commands. Do not use the st analysis commands unless you want ridiculous results, but one of the st data management commands is just what we want:

```
. stfill sex, forward

          failure _d:  code == -1
    analysis time _t:  (date-origin)
              origin:  min
                  id:  patid

replace missing values with previously observed values:
              sex:  203 real changes made
```

Problem one solved.

The second problem concerns the lack of an admission-date variable. That is not really a problem because we have a new first observation with `code` = 22 recording the date of admission, so every place we previously coded `origin(time addate)`, we substitute `origin(code == 22)`.

Problem two solved.

We also solved the big problem—converting a snapshot dataset into a survival-time dataset; see [ST] **snapspan**.

◁

Two concepts of time

The st system has two concepts of time. The first is *time* in italics, which corresponds to how time is recorded in your data. The second is analysis time, which we write as t. Substantively, analysis time is time at risk. `stset` defines analysis time in terms of *time* via

$$t = \frac{\mathit{time} - \texttt{origin()}}{\texttt{scale()}}$$

t and *time* can be the same thing, and by default they are because, by default, `origin()` is 0 and `scale()` is 1.

> All the st analysis commands work with analysis time, t.
>
> By default, if you do not specify the `origin()` and `scale()` options, your time variables are expected to be the analysis-time variables. This means that $\mathit{time} = 0$ corresponds to when subjects became at risk, and that means, among other things, that observations for which $\mathit{time} < 0$ are ignored because survival analysis concerns persons who are at risk, and no one is at risk before $t = 0$.
>
> `origin` determines when the clock starts ticking. If you do not specify `origin()`, `origin(time 0)` is assumed, meaning that $t = \mathit{time}$ and that persons are at risk from $t = \mathit{time} = 0$.

time and t will often differ. *time* might be calendar time and t the length of time since some event, such as being born or being exposed to some risk factor. `origin()` sets when $t = 0$. `scale()` merely sets a constant that makes t more readable.

The syntax for the `origin()` option makes it look more complicated than it really is

origin([*varname* == *numlist*] time *exp* | min)

This says that there are four different ways to specify `origin()`:

```
origin(time exp)
origin(varname == numlist)
origin(varname == numlist time exp)
origin(min)
```

The first syntax can be used with single- or multiple-record data. It states that the origin is given by *exp*, which can be a constant for all observations, a variable (and hence may vary subject by subject), or even an expression composed of variables and constants. Perhaps the origin is a fixed date or a date recorded in the data when the subject was exposed or when the subject turned 18.

The second and third syntaxes are for use with multiple-record data. The second states that the origin corresponds to the (earliest) time when the designated event occurred. Perhaps the origin is when an operation was performed. The third syntax calculates the origin both ways and then selects the later one.

The fourth syntax does something odd; it sets the origin to the minimum time observed minus 1. This is not useful for analysis but is sometimes useful for playing data management tricks; see example 10 above.

Let's start with the first syntax. Say that you had the data

```
 faildate     x1      x2
28dec1997     12      22
12nov1997     15      22
03feb1998     55      22
```

and that all the observations came at risk on the same date, 01nov1997. You could type

```
. stset faildate, origin(time mdy(11,1,1997))
```

Remember that `stset` adds `_t0` and `_t` to your dataset and that they contain the time span for each record, documented in analysis-time units. After typing `stset`, you can list the results:

```
. list faildate x1 x2 _t0 _t

     +-------------------------------------+
     |  faildate   x1   x2   _t0      _t  |
     |-------------------------------------|
  1. | 28dec1997   12   22     0      57  |
  2. | 12nov1997   15   22     0      11  |
  3. | 03feb1998   55   22     0      94  |
     +-------------------------------------+
```

Record 1 reflects the period $(0, 57]$ in analysis-time units, which are days here. `stset` calculated the 57 from 28dec1997 − 01nov1997 = 13,876 − 13,819 = 57. (Dates such as 28dec1997 are really just integers containing the number of days from 01jan1960, and Stata's `%td` display format makes them display nicely. 28dec1997 is really the number 13,876.)

As another example, we might have data recording exposure and failure dates:

```
  expdate    faildate       x1       x2
07may1998   22jun1998       12       22
02feb1998   11may1998       11       17
```

The way to stset this dataset is

```
      . stset faildate, origin(time expdate)
```

and the result, in analysis units, is

```
      . list expdate faildate x1 x2 _t0 _t

             ┌──────────────────────────────────────────────┐
             │   expdate    faildate   x1   x2   _t0   _t   │
             ├──────────────────────────────────────────────┤
          1. │ 07may1998   22jun1998   12   22     0   46   │
          2. │ 02feb1998   11may1998   11   17     0   98   │
             └──────────────────────────────────────────────┘
```

There is nothing magical about dates. Our original data could just as well have been

```
             expdate   faildate      x1       x2
                  32         78      12       22
                  12        110      11       17
```

and the result would still be the same because $78 - 32 = 46$ and $110 - 12 = 98$.

Specifying an expression can sometimes be useful. Suppose that your dataset has the variable date recording the date of event and variable age recording the subject's age as of date. You want to make $t = 0$ correspond to when the subject turned 18. You could type origin(time date-int((age-18)*365.25)).

origin(*varname* == *numlist*) is for use with multiple-record data. It states when each subject became at risk indirectly; the subject became at risk at the earliest time that *varname* takes on any of the enumerated values. Say that you had

```
          patid       date      x1      x2    event
            101  12nov1997      15      22      127
            101  28dec1997      12      22      155
            101  03feb1998      55      22      133
            101  05mar1998      14      22      127
            101  09apr1998      12      22      133
            101  03jun1998      13      22      101
            102  22nov1997       .       .        .
```

and assume event $= 155$ represents the onset of exposure. You might stset this dataset by typing

```
      . stset date, id(patid) origin(event==155) ...
```

If you did that, the information for patient 101 before 28dec1997 would be ignored in subsequent analysis. The prior information would not be removed from the dataset; it would just be ignored. Probably something similar would happen for patient 102, or if patient 102 has no record with event $= 155$, all the records on the patient would be ignored.

For analysis time, $t = 0$ would correspond to when event 155 occurred. Here are the results in analysis-time units:

```
          patid       date      x1      x2    event           _t0           _t
            101  12nov1997      15      22      127             .            .
            101  28dec1997      12      22      155             .            .
            101  03feb1998      55      22      133             0           37
            101  05mar1998      14      22      127            37           67
            101  09apr1998      12      22      133            67          102
            101  03jun1998      13      22      101           102          157
            102  22nov1997       .       .        .             .            .
```

Patient 101's second record is excluded from the analysis. That is not a mistake. Records document durations, `date` reflects the end of the period, and events occur at the end of periods. Thus event 155 occurred at the instant `date` = 28dec1997, and the relevant first record for the patient is $(28\text{dec}1997, 03\text{feb}1998]$ in time units, which is $(0, 37\,]$ in t units.

The substantive meaning of analysis time

In specifying `origin()`, you must ask yourself whether two subjects with identical characteristics face the same risk of failure. The answer is that they face the same risk when they have the same value of $t = (\textit{time} - \texttt{origin()})/\texttt{scale()}$ or, equivalently, when the same amount of time has elapsed from `origin()`.

Say that we have the following data on smokers who have died:

```
     ddate       x1       x2   reason
11mar1984       23       11        2
15may1994       21        9        1
22nov1993       22       13        2
etc.
```

We wish to analyze death due to `reason==2`. However, typing

```
. stset ddate, fail(reason==2)
```

would probably not be adequate. We would be saying that smokers were at risk of death from 01jan1960. Would it matter? It would if we planned on doing anything parametric because parametric hazard functions, except for the exponential, are functions of analysis time, and the location of 0 makes a difference.

Even if we were thinking of performing nonparametric analysis, there would probably be difficulties. We would be asserting that two "identical" persons (in terms of `x1` and `x2`) face the same risk on the same calendar date. Does the risk of death due to smoking really change as the calendar changes?

It would be more reasonable to assume that the risk changes with how long a subject has been smoking and that our data would probably include that date. We would type

```
. stset ddate, fail(reason==2) origin(time smdate)
```

if `smdate` were the name of the date-started-smoking variable. We would now be saying that the risk is equal when the number of days smoked is the same. We might prefer to see t in years,

```
. stset ddate, fail(reason==2) origin(time smdate) scale(365.25)
```

but that would make no substantive difference.

Consider single-record data on firms that went bankrupt:

```
   incorp   bankrupt      x1      x2  btype
22jan1983  11mar1984      23      11      2
17may1992  15may1994      21       9      1
03nov1991  22nov1993      22      13      2
etc.
```

Say that we wish to examine the risk of a particular kind of bankruptcy, `btype == 2`, among firms that become bankrupt. Typing

```
. stset bankrupt, fail(btype==2)
```

would be more reasonable than it was in the smoking example. It would not be reasonable if we were thinking of performing any sort of parametric analysis, of course, because then location of $t = 0$ would matter, but it might be reasonable for semiparametric analysis. We would be asserting that two "identical" firms (with respect to the characteristics we model) have the same risk of bankruptcy when the calendar dates are the same. We would be asserting that the overall state of the economy matters.

It might be reasonable to instead measure time from the date of incorporation:

```
. stset bankrupt, fail(btype==2) origin(time incorp)
```

Understand that the choice of `origin()` is a substantive decision.

Setting the failure event

You set the failure event by using the `failure()` option.

In single-record data, if `failure()` is not specified, every record is assumed to end in a failure. For instance, with

	failtime	load	bearings
1.	100	15	0
2.	140	15	1

etc.

you would type `stset failtime`, and the first observation would be assumed to fail at time = 100; the second, at time = 140; and so on.

`failure(`*varname*`)` specifies that a failure occurs whenever *varname* is not zero and is not missing. For instance, with

	failtime	load	bearings	burnout
1.	100	15	0	1
2.	140	15	1	0
3.	97	20	0	1
4.	122	20	1	0
5.	84	25	0	1
6.	100	25	1	1

etc.

you might type `stset failtime, failure(burnout)`. Observations 1, 3, 5, and 6 would be assumed to fail at times 100, 97, 84, and 100, respectively; observations 2 and 4 would be assumed to be censored at times 140 and 122.

Similarly, if the data were

	failtime	load	bearings	burnout
1.	100	15	0	1
2.	140	15	1	0
3.	97	20	0	2
4.	122	20	1	.
5.	84	25	0	2
6.	100	25	1	3

etc.

the result would be the same. Nonzero, nonmissing values of the failure variable are assumed to represent failures. (Perhaps `burnout` contains a code on how the burnout occurred.)

`failure(`*varname* `==` *numlist*`)` specifies that a failure occurs whenever *varname* takes on any of the values of *numlist*. In the above example, specifying

```
. stset failtime, failure(burnout==1 2)
```

would treat observation 6 as censored.

```
    . stset failtime, failure(burnout==1 2 .)
```

would also treat observation 4 as a failure.

```
    . stset failtime, failure(burnout==1/3 6 .)
```

would treat `burnout==1`, `burnout==2`, `burnout==3`, `burnout==6`, and `burnout==.` as representing failures and all other values as representing censorings. (Perhaps we want to examine “failure due to meltdown”, and these are the codes that represent the various kinds of meltdown.)

`failure()` is treated the same way in both single- and multiple-record data. Consider

```
          patno        t       x1       x2     died
     1.       1        4       23       11        1
     2.       2        5       21        9        0
     3.       2        8       22       13        1
     4.       3        7       20        5        0
     5.       3        9       22        5        0
     6.       3       11       21        5        0
     7.       4      ...
```

Typing

```
    . stset t, id(patno) failure(died)
```

would treat

`patno==1`	as dying	at `t==4`
`patno==2`	as dying	at `t==8`
`patno==3`	as being censored	at `t==11`

Intervening records on the same subject are marked as “censored”. Technically, they are not really censored if you think about it carefully; they are simply marked as not failing. Look at the data for subject 3:

```
          patno        t       x1       x2     died
              3        9       22        5        0
              3       11       21        5        0
```

The subject is not censored at `t` = 9 because there are more data on the subject; it is merely the case that the subject did not die at that time. At `t` = 9, `x1` changed from 22 to 21. The subject is really censored at `t` = 11 because the subject did not die and there are no more records on the subject.

Typing `stset t, id(patno) failure(died)` would mark the same persons as dying and the same persons as censored, as in the previous case. If `died` contained not 0 and 1, but 0 and nonzero, nonmissing codes for the reason for death would be

```
          patno        t       x1       x2     died
     1.       1        4       23       11      103
     2.       2        5       21        9        0
     3.       2        8       22       13      207
     4.       3        7       20        5        0
     5.       3        9       22        5        0
     6.       3       11       21        5        0
     7.       4      ...
```

Typing

```
    . stset t, id(patno) failure(died)
```

or

```
    . stset t, id(patno) failure(died==103 207)
```

would yield the same results; subjects 1 and 2 would be treated as dying and subject 3 as censored.

Typing

```
. stset t, id(patno) failure(died==207)
```

would treat subject 2 as dying and subjects 1 and 3 as censored. Thus when you specify the values for the code, the code variable need not ever contain 0. In

```
        patno       t      x1      x2    died
  1.        1       4      23      11     103
  2.        2       5      21       9      13
  3.        2       8      22      13     207
  4.        3       7      20       5      11
  5.        3       9      22       5      12
  6.        3      11      21       5      12
  7.        4     ...
```

typing

```
. stset t, id(patno) failure(died==207)
```

treats patient 2 as dying and 1 and 3 as censored. Typing

```
. stset t, id(patno) failure(died==103 207)
```

treats patients 1 and 2 as dying and 3 as censored.

Setting multiple failures

In multiple-record data, records after the first failure event are ignored unless you specify the `exit()` option. Consider the following data:

```
        patno       t      x1      x2    code
  1.        1       4      21       7      14
  2.        1       5      21       7      11
  3.        1       7      20       7      17
  4.        1       8      22       7      22
  5.        1       9      22       7      22
  6.        1      11      21       7      29
  7.        2     ...
```

Perhaps code 22 represents the event of interest—say, the event "visited the doctor". Were you to type `stset t, id(patno) failure(code == 22)`, the result would be as if the data contained

```
        patno       t      x1      x2    code
  1.        1       4      21       7      14
  2.        1       5      21       7      11
  3.        1       7      20       7      17
  4.        1       8      22       7      22
```

Records after the first occurrence of the failure event are ignored. If you do not want this, you must specify the `exit()` option. Probably you would want to specify `exit(time .)`, here meaning that subjects are not to exit the risk group until their data run out. Alternatively, perhaps code 142 means "entered the nursing home" and, once that event happens, you no longer want them in the risk group. Then you would code `exit(code == 142)`; see *Final exit times* below.

First entry times

Do not confuse `enter()` with `origin()`. `origin()` specifies when a subject first becomes at risk. `enter()` specifies when a subject first comes under observation. In most datasets, becoming at risk and coming under observation are coincident. Then it is sufficient to specify `origin()` alone, although you could specify both options.

Some persons enter the data after they have been at risk of failure. Say that we are studying deaths due to exposure to substance X and we know the date at which a person was first exposed to the substance. We are willing to assume that persons are at risk from the date of exposure forward. A person arrives at our door who was exposed 15 years ago. Can we add this person to our data? The statistical issue is labeled *left-truncation*, and the problem is that had the person died before arriving at our door, we would never have known about her. We can add her to our data, but we must be careful to treat her subsequent survival time as conditional on having already survived 15 years.

Say that we are examining visits to the widget repair facility, "failure" being defined as a visit (so failures can be repeated). The risk begins once a person buys a widget. We have a woman who bought a widget 3 years ago, and she has no records on when she has visited the facility in the last 3 years. Can we add her to our data? Yes, as long as we are careful to treat her subsequent behavior as already being 3 years after she first became at risk.

The jargon for this is "under observation". All this means is that any failures would be observed. Before being under observation, failure would not be observed.

If `enter()` is not specified, we assume that subjects are under observation at the time they enter the risk group as specified by `origin()`, 0 if `origin()` is not specified, or possibly `time0()`. To be precise, subject i is assumed to first enter the analysis risk pool at

$$\mathit{time}_i = \max\Big(\text{earliest }\texttt{time0()}\text{ for } i, \texttt{enter()}, \texttt{origin()}\Big)$$

Say that we have multiple-record data recording "came at risk" (`mycode == 1`), "enrolled in our study" (`mycode == 2`), and "failed due to risk" (`mycode == 3`). We `stset` this dataset by typing

```
. stset time, id(id) origin(mycode==1) enter(mycode==2) failure(mycode==3)
```

The above `stset` correctly handles the came at risk/came under observation problem regardless of the order of events 1, 2, and 3. For instance, if the subject comes under observation before he or she becomes at risk, the subject will be treated as entering the analysis risk pool at the time he or she came at risk.

Say that we have the same data in single-record format: variable `riskdate` documents becoming at risk and variable `enr_date` the date of enrollment in our study. We would `stset` this dataset by typing

```
. stset time, origin(time riskdate) enter(time enr_date) failure(mycode==3)
```

For a final example, let's return to the multiple-record way of recording our data and say that we started enrolling people in our study on 12jan1998 but that, up until 16feb1998, we do not trust that our records are complete (we had start-up problems). We would `stset` that dataset by typing

```
. stset time, origin(mycode==1) enter(mycode==2 time mdy(2,16,1998))
> fail(mycode==3)
```

`enter(`*varname*==*numlist* `time` *exp*`)` is interpreted as

$$\max(\text{time of earliest event in } \mathit{numlist}, \mathit{exp})$$

Thus persons having `mycode == 2` occurring before 16feb1998 are assumed to be under observation from 16feb1998, and those having `mycode == 2` thereafter are assumed to be under observation from the time of `mycode == 2`.

Final exit times

`exit()` specifies the latest time under which the subject is both under observation and at risk of the failure event. The emphasis is on latest; obviously subjects also exit the data when their data run out.

When you type

```
. stset ..., ... failure(outcome==1/3 5) ...
```

the result is as if you had typed

```
. stset ..., ... failure(outcome==1/3 5) exit(failure) ...
```

which, in turn, is the same as

```
. stset ..., ... failure(outcome==1/3 5) exit(outcome==1/3 5) ...
```

When are people to be removed from the analysis risk pool? When their data end, of course, and when the event 1, 2, 3, or 5 first occurs. How are they to be removed? According to their status at that time. If the event is 1, 2, 3, or 5 at that instant, they exit as a failure. If the event is something else, they exit as censored.

Perhaps events 1, 2, 3, and 5 represent death due to heart disease, and that is what we are studying. Say that `outcome == 99` represents death for some other reason. Obviously, once the person dies, she is no longer at risk of dying from heart disease, so we would want to specify

```
. stset ..., ... failure(outcome==1/3 5) exit(outcome==1/3 5 99) ...
```

When we explicitly specify `exit()`, we must list all the reasons for which a person is to be removed other than simply running out of data. Here it would have been a mistake to specify just `exit(99)` because that would have left persons in the analysis risk pool who died for reasons 1, 2, 3, and 5. We would have treated those people as if they were still at risk of dying.

In fact, it probably would not have mattered had we specified `exit(99)` because, once a person is dead, he or she is unlikely to have any subsequent records anyway. By that logic, we did not even have to specify `exit(99)` because death is death and there should be no records following it.

For other kinds of events, however, `exit()` becomes important. Let's assume that the failure event is to be diagnosed with heart disease. A person may surely have records following diagnosis, but even so,

```
. stset ..., ... failure(outcome==22) ...
```

would be adequate because, by not specifying `exit()`, we are accepting the default that `exit()` is equivalent to `failure()`. Once outcome 22 occurs, subsequent records on the subject will be ignored—they constitute the future history of the subject.

Say, however, that we wish to treat as censored persons diagnosed with kidney disease. We would type

```
. stset ..., ... failure(outcome==22) exit(outcome==22 29) ...
```

assuming that `outcome` = 29 is "diagnosed with kidney disease". It is now of great importance that we specified `exit(outcome==22 29)` and not just `exit(outcome==29)` because, had we omitted code 22, persons would have remained in the analysis risk pool even after the failure event, that is, being diagnosed with heart disease.

If, in addition, our data were untrustworthy after 22nov1998 (perhaps not all the data have been entered yet), we would type

```
. stset ..., ... failure(outcome==22) exit(outcome==22 29 time
> mdy(11,22,1998)) ...
```

If we type `exit(`*varname*`==`*numlist* `time` *exp*`)`, the exit time is taken to be

$$\min(\text{time of earliest event in } numlist, exp)$$

For some analyses, repeated failures are possible. If you have repeated failure data, you specify the `exit()` option and include whatever reasons, if any, that would cause the person to be removed. If there are no such reasons and you wish to retain all observations for the person, you type

```
. stset ..., ... exit(time .) ...
```

`exit(time .)` specifies that the maximum time a person can be in the risk pool is infinite; thus subjects will not be removed until their data run out.

Intermediate exit and reentry times (gaps)

Gaps arise when a subject is temporarily not under observation. The statistical importance of gaps is that, if failure is death and if the person died during such a gap, he would not have been around to be found again. The solution to this is to remove the person from the risk pool during the observational gap.

To determine that you have gaps, your data must provide starting and ending times for each record. Most datasets provide only ending times, making it impossible to know that you have gaps.

You use `time0()` to specify the beginning times of records. `time0()` specifies a mechanical aspect of interpreting the records in the dataset, namely, the beginning of the period spanned by each record. Do not confuse `time0()` with `origin()`, which specifies when a subject became at risk, or with `enter()`, which specifies when a subject first comes under observation.

`time0()` merely identifies the beginning of the time span covered by each record. Say that we had two records on a subject, the first covering the span (40,49] and the second, (49,57]:

```
                |<—— record 1 ——>|<- record 2 ->|
        ————————|————————————————|——————————————|——> time
               40               49             57
```

A `time0()` variable would contain

40 in record 1
49 in record 2

and not, for instance, 40 and 40. A `time0()` variable varies record by record for a subject.

Most datasets merely provide an end-of-record time value, *timevar*, which you specify by typing `stset` *timevar*`, ...`. When you have multiple records per subject and you do not specify a `time0()` variable, `stset` assumes that each record begins where the previous one left off.

if() versus if exp

Both the `if` *exp* and `if(`*exp*`)` options select records for which *exp* is true. We strongly recommend specifying the `if()` option in preference to `if` *exp*. They differ in that `if` *exp* removes data from consideration before calculating beginning and ending times, and other quantities as well. The `if()` option, on the other hand, sets the restriction after all derived variables are calculated. To appreciate this difference, consider the following multiple-record data:

```
    patno         t        x1        x2      code
        3         7        20         5        14
        3         9        22         5        23
        3        11        21         5        29
```

Consider the difference in results between typing

```
. stset t if x1!=22, failure(code==14)
```

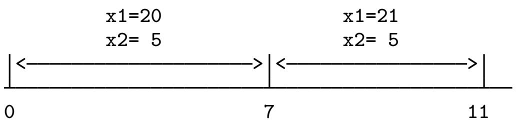
and

```
. stset t,  if(x1!=22) failure(code==14)
```

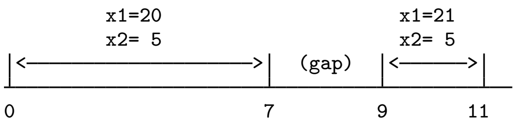
The first would remove record 2 from consideration at the outset. In constructing beginning and ending times, `stset` and `streset` would see

```
patno       t      x1      x2    code
    3       7      20       5      14
    3      11      21       5      29
```

and would construct the result:



In the second case, the result would be



The latter result is correct and the former incorrect because `x1` $= 21$ is not true in the interval $(7, 9)$.

The only reason to specify `if` *exp* is to ignore errors in the data—observations that would confuse `stset` and `streset`—without actually dropping the offending observations from the dataset.

You specify the `if()` option to ignore information in the data that are not themselves errors. Specifying `if()` yields the same result as specifying `if` *exp* on the subsequent st commands after the dataset has been `stset`.

Past and future records

Consider the hospital ward data that we have seen before:

```
patid     addate     curdate     sex      x1      x2    code
  101  18aug1998   23aug1998       1      10      10     177
  101          .   31aug1998       1      20       8     286
  101          .   08sep1998       1      16      11     208
  101          .   11sep1998       1      11      17     401
  etc.
```

Say that you `stset` this dataset such that you selected the middle two records. Perhaps you typed

```
. stset curdate, id(patid) origin(time addate) enter(code==286) failure(code==208)
```

The first record for the subject, because it was not selected, is called a *past history record.* Any earlier records that were not selected would also be called past history records.

The last record for the subject, because it was not selected, is called a *future history record.* Any later records that were not selected would also be called future history records.

If you typed

```
. streset, past
```

the first three records for this subject would be selected.

Typing

```
. streset, future
```

would select the last three records for this subject.

If you typed

```
. streset, past future
```

all four records for this subject would be selected.

If you then typed

```
. streset
```

the original two records would be selected, and things would be back just as they were before.

After typing `streset, past`; `streset, future`; or `streset, past future`, you would not want to use any analysis commands. `streset` did some strange things, especially with the analysis-time variable, to include the extra records. It would be the wrong sample, anyway.

You might, however, want to use certain data management commands on the data, especially those for creating new variables.

Typically, `streset, past` is of greater interest than the other commands. Past records—records prior to being at risk or excluded for other reasons—are not supposed to play a role in survival analysis. `stset` makes sure they do not. But it is sometimes reasonable to ask questions about them such as, was the subject ever on the drug cisplatin? Has the subject ever been married? Did the subject ever have a heart attack?

To answer questions like that, you sometimes want to dig into the past. Typing `streset, past` makes that easy, and once the past is set, the data can be used with `stgen` and `st_is 2`. You might well type the following:

```
. stset curdate, id(patid) origin(addate) enter(code==286) failure(code==208)
. streset, past
. stgen attack = ever(code==177)
. streset
. stcox attack ...
```

Do not be concerned about doing something inappropriate while having the past or future set; st will not let you:

```
. stset curdate, id(patid) origin(time addate) enter(code==286) failure(code==208)
  (output omitted)
. streset, past
  (output omitted)
. stcox x1
you last streset, past
you must type streset to restore the analysis sample
r(119);
```

Using streset

streset is a useful tool for gently modifying what you have previously stset. Rather than typing the whole stset command, you can type streset followed just by what has changed.

For instance, you might type

```
. stset curdate, id(patid) origin(time addate) enter(code==286) failure(code==208)
```

and then later want to restrict the analysis to subjects who ever have x1>20. You could retype the whole stset command and add ever(x1>20), but it would be easier to type

```
. streset, ever(x1>20)
```

If later you decide you want to remove the restriction, type

```
. streset, ever(.)
```

That is the general rule for resetting options to the default: type '.' as the option's argument.

Be careful using streset because you can make subtle mistakes. In another analysis with another dataset, consider the following:

```
. stset date, fail(code==2) origin(code==1107)
. ...
. streset date, fail(code==9) origin(code==1422) after(code==1423)
. ...
. streset, fail(code==2) origin(code==1107)
```

If, in the last step, you are trying to get back to the results of the first stset, you will fail. The last streset is equivalent to

```
. stset date, fail(code==9) origin(code==1107) after(code==1423)
```

streset() remembers the previously specified options and uses them if you do not override them. Both stset and streset display the current command line. Make sure that you verify that the command is what you intended.

Performance and multiple-record-per-subject datasets

stset and streset do not drop data; they simply mark data to be excluded from consideration. Some survival-time datasets can be large, although the relevant subsamples are small. In such cases, you can reduce memory requirements and speed execution by dropping the irrelevant observations.

stset and streset mark the relevant observations by creating a variable named _st (it is always named this) containing 1 and 0; _st = 1 marks the relevant observations and _st = 0 marks the irrelevant ones. If you type

```
. drop if _st==0
```

or equivalently

```
. keep if _st==1
```

or equivalently

```
. keep if _st
```

you will drop the irrelevant observations. All st commands produce the same results whether you do this or not. Be careful, however, if you are planning future stsets or stresets. Observations that are irrelevant right now might be relevant later.

One solution to this conundrum is to keep only those observations that are relevant after setting the entire history:

```
. stset date, fail(code==9) origin(code==1422) after(code==1423)
. streset, past future
. keep if _st
. streset
```

As a final note, you may drop the irrelevant observations as marked by `_st` = 0, but do not drop the `_st` variable itself. The other st commands expect to find variable `_st`.

Sequencing of events within t

Consider the following bit of data:

```
etime   failtime       fail
    0          5          1
    0          5          0
    5          7          1
```

Note all the different events happening at time 5: the first observation fails, the second is censored, and the third enters.

What does it mean for something to happen at time 5? In particular, is it at least potentially possible for the second observation to have failed at time 5; that is, was it in the risk group when the first observation failed? How about the third observation? Was it in the risk group, and could it have potentially failed at time 5?

Stata sequences events within a time as follows:

first,	at time	t	the failures occur
then,	at time	$t+0$	the censorings are removed from the risk group
finally,	at time	$t+0+0$	the new entries are added to the risk group

Thus, to answer the questions:

Could the second observation have potentially failed at time 5? Yes.

Could the third observation have potentially failed at time 5? No, because it was not yet in the risk group.

By this logic, the following makes no sense:

```
etime   failtime       fail
    5          5          1
```

This would mark a subject as failing before being at risk. It would make no difference if `fail` were 0—the subject would then be marked as being censored too soon. Either way, `stset` would flag this as an error. If you had a subject who entered and immediately exited, you would code this as

```
etime   failtime       fail
 4.99          5          1
```

Weights

`stset` allows you to specify `fweights`, `pweights`, and `iweights`.

`fweights` are Stata's frequency or replication weights. Consider the data

```
        failtime       load   bearings       count
             100         15          0           3
             140         15          1           2
              97         20          0           1
```

and the `stset` command

```
. stset failtime [fw=count]

     failure event:  (assumed to fail at time=failtime)
obs. time interval:  (0, failtime]
 exit on or before:  failure
            weight:  [fweight=count]

──────────────────────────────────────────────────────────────────────────────
        3  total observations
        0  exclusions
──────────────────────────────────────────────────────────────────────────────
        3  physical observations remaining, equal to
        6  weighted observations, representing
        6  failures in single-record/single-failure data
      677  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =       140
```

This combination is equivalent to the expanded data

```
        failtime       load   bearings
             100         15          0
             100         15          0
             100         15          0
             140         15          1
             140         15          1
              97         20          0
```

and the command

```
. stset failtime
```

`pweights` are Stata's sampling weights—the inverse of the probability that the subject was chosen from the population. `pweights` are typically integers, but they do not have to be. For instance, you might have

```
           time0       time       died         sex        reps
               0        300          1           0        1.50
               0        250          0           1        4.50
              30        147          1           0        2.25
```

Here `reps` is how many patients each observation represents in the underlying population—perhaps when multiplied by 10. The `stset` command for these data is

```
. stset time [pw=reps], origin(time time0) failure(died)
```

For variance calculations, the scale of the `pweights` does not matter. `reps` in the 3 observations shown could just as well be 3, 9, and 4.5. Nevertheless, the scale of the `pweights` is used when you ask for counts. For instance, `stsum` would report the person-time at risk as

$$(300 - 0)\,1.5 + (250 - 0)\,4.5 + (147 - 30)\,2.25 = 1{,}838.25$$

for the 3 observations shown. `stsum` would count that $1.5 + 2.25 = 3.75$ persons died, and so the incidence rate for these 3 observations would be $3.75/1{,}838.25 = 0.0020$. The incidence rate is thus unaffected by the scale of the weights. Similarly, the coefficients and confidence intervals reported by, for instance, `streg, dist(exponential)` would be unaffected. The 95% confidence interval for the incidence rate would be $[\,0.0003, 0.0132\,]$, regardless of the scale of the weights.

If these 3 observations were examined unweighted, the incidence rate would be 0.0030 and the 95% confidence interval would be $[\,0.0007, 0.0120\,]$.

Finally, `stset` allows you to set `iweights`, which are Stata's "importance" weights, but we recommend that you do not. `iweights` are provided for those who wish to create special effects by manipulating standard formulas. The st commands treat `iweights` just as they would `fweights`, although they do not require that the weights be integers, and push their way through conventional variance calculations. Thus results—counts, rates, and variances—depend on the scale of these weights.

Data warnings and errors flagged by stset

When you `stset` your data, `stset` runs various checks to verify that what you are setting makes sense. `stset` refuses to set the data only if, in multiple-record, weighted data, weights are not constant within ID. Otherwise, `stset` merely warns you about any inconsistencies that it identifies.

Although `stset` will set the data, it will mark out records that it cannot understand; for instance,

```
. stset curdate, origin(time addate) failure(code==402) id(patid)

                id:  patid
     failure event:  code == 402
obs. time interval:  (curdate[_n-1], curdate]
 exit on or before:  failure
    t for analysis:  (time-origin)
            origin:  time addate

------------------------------------------------------------------------------
      243  total observations
        1  event time missing (curdate>=.)                     PROBABLE ERROR
        4  multiple records at same instant                    PROBABLE ERROR
           (curdate[_n-1]==curdate)
------------------------------------------------------------------------------
      238  observations remaining, representing
       40  subjects
       15  failures in single failure-per-subject data
     1478  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =        62
```

You must ensure that the result, after exclusions, is correct.

The warnings `stset` might issue include

```
ignored because patid missing
event time missing                                PROBABLE ERROR
entry time missing                                PROBABLE ERROR
entry on or after exit (etime>t)                  PROBABLE ERROR
obs. end on or before enter()
obs. end on or before origin()
multiple records at same instant (t[_n-1]==t)     PROBABLE ERROR
overlapping records (t[_n-1]>entry time)          PROBABLE ERROR
weights invalid                                   PROBABLE ERROR
```

`stset` sets `_st` = 0 when observations are excluded for whatever reason. Thus observations with any of the above problems can be found among the `_st` = 0 observations.

Using survival-time data in Stata

In the examples above, we have shown you how Stata expects survival-time data to be recorded. To summarize:

- Each subject's history is represented by 1 or more observations in the dataset.
- Each observation documents a span of time. The observation must contain when the span ends (exit time) and may optionally contain when the span begins (entry time). If the entry time is not recorded, it is assumed to be 0 or, in multiple-record data, the exit time of the subject's previous observation, if there is one. By *previous*, we mean that the data have already been temporally ordered on exit times within subject. The physical order of the observations in your dataset does not matter.
- Each observation documents an outcome associated with the exit time. Unless otherwise specified with `failure()`, 0 and missing mean censored, and nonzero means failed.
- Each observation contains other variables (called covariates) that are assumed to be constant over the span of time recorded by the observation.

Data rarely arrive in this neatly organized form. For instance, Kalbfleisch and Prentice (2002, 4–5) present heart transplant survival data from Stanford (Crowley and Hu 1977). These data can be converted into the correct st format in at least two ways. The first method is shown in example 11. A second, shorter, method using the st commands is described in example 3 of [ST] **stsplit**.

▷ Example 11

Here we will describe the process that uses the standard Stata commands.

```
. use http://www.stata-press.com/data/r13/stan2, clear
(Heart transplant data)

. describe

Contains data from http://www.stata-press.com/data/r13/stan2.dta
  obs:           103                          Heart transplant data
 vars:             5                          30 Nov 2012 11:14
 size:         1,030
--------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
id              int     %8.0g                 Patient Identifier
died            byte    %8.0g                 Survival Status (1=dead)
stime           float   %8.0g                 Survival Time (Days)
transplant      byte    %8.0g                 Heart Transplant
wait            int     %8.0g                 Waiting Time
--------------------------------------------------------------------------------
Sorted by:
```

The data are from 103 patients selected as transplantation candidates. There is one record on each patient, and the important variables, from an st-command perspective, are

`id`	the patient's ID number
`transplant`	whether the patient received a transplant
`wait`	when (after acceptance) the patient received the transplant
`stime`	when (after acceptance) the patient died or was censored
`died`	the patient's status at `stime`

To better understand, let's examine two records from this dataset:

```
. list id transplant wait stime died if id==44 | id==16

      ┌──────────────────────────────────────────┐
      │ id   transp~t   wait   stime   died      │
      ├──────────────────────────────────────────┤
 33.  │ 44          0      0      40      1      │
 71.  │ 16          1     28     308      1      │
      └──────────────────────────────────────────┘
```

Patient 44 never did receive a new heart; he or she died 40 days after acceptance while still on the waiting list. Patient 16 did receive a new heart—28 days after acceptance—yet died 308 days after acceptance.

Our goal is to turn this into st data that contain the histories of each of these patients. That is, we want records that appear as

```
id       t1      died  posttran
16       28         0         0
16      308         1         1
44       40         1         0
```

or, even more explicitly, as

```
id         t0         t1      died  posttran
16          0         28         0         0
16         28        308         1         1
44          0         40         1         0
```

The new variable, `posttran`, would be 0 before transplantation and 1 afterward.

Patient 44 would have one record in this new dataset recording that he or she died at time 40 and that `posttran` was 0 over the entire interval.

Patient 16, however, would have two records: one documenting the duration $(0, 28\,]$, during which `posttran` was 0, and one documenting the duration $(28, 308\,]$, during which `posttran` was 1.

Our goal is to take the first dataset and convert it into the second, which we can then `stset`. We make the transformation by using Stata's other data management commands. One way we could do this is by typing

```
. expand 2 if transplant
(69 observations created)

. by id, sort: gen byte posttran = (_n==2)

. by id: gen t1 = stime if _n==_N
(69 missing values generated)

. by id: replace t1 = wait if _n==1 & transplant
(69 real changes made)

. by id: replace died=0 if _n==1 & transplant
(45 real changes made)
```

`expand 2 if transplant` duplicated the observations for patients who had `transplant` $\neq$ 0. Considering our two sample patients, we would now have the following data:

```
id  transp~t      wait     stime       died
44         0         0        40          1
16         1        28       308          1
16         1        28       308          1
```

We would have 1 observation for patient 44 and 2 identical observations for patient 16.

We then `by id, sort: gen posttran = (_n==2)`, resulting in

```
               id  transp~t      wait     stime      died  posttran
               16         1        28       308         1         0
               16         1        28       308         1         1
               44         0         0        40         1         0
```

This type of trickiness is discussed in [U] **13.7 Explicit subscripting**. Statements such as `_n==2` produce values 1 (meaning true) and 0 (meaning false), so new variable `posttran` will contain 1 or 0 depending on whether `_n` is or is not 2. `_n` is the observation counter and, combined with `by id:`, becomes the observation-within-ID counter. Thus we set `posttran` to 1 on second records and to 0 on all first records.

Finally, we produce the exit-time variable. Final exit time is just `stime`, and that is handled by the command `by id: gen t1 = stime if _n==_N`. `_n` is the observation-within-ID counter and `_N` is the total number of observations within `id`, so we just set the last observation on each patient to `stime`. Now we have

```
               id  transp~t      wait     stime      died  posttran        t1
               16         1        28       308         1         0         .
               16         1        28       308         1         1       308
               44         0         0        40         1         0        40
```

All that is left to do is to fill in `t1` with the value from `wait` on the interim records, meaning `replace t1=wait if` it is an interim record.

There are many ways we could identify the interim records. In the output above, we did it by

```
    . by id: replace t1 = wait if _n==1 & transplant
```

meaning that an interim record is a first record of a person who did receive a transplant. More easily, but with more trickery, we could have just said

```
    . replace t1=wait if t1>=.
```

because the only values of `t1` left to be filled in are the missing ones. Another alternative would be

```
    . by id: replace t1 = wait if _n==1 & _N==2
```

which would identify the first record of two-record pairs. There are many alternatives, but they would all produce the same thing:

```
               id  transp~t      wait     stime      died  posttran        t1
               16         1        28       308         1         0        28
               16         1        28       308         1         1       308
               44         0         0        40         1         0        40
```

There is one more thing we must do, which is to reset `died` to contain 0 on the interim records:

```
    . by id: replace died=0 if _n==1 & transplant
```

The result is

```
               id  transp~t      wait     stime      died  posttran        t1
               16         1        28       308         0         0        28
               16         1        28       308         1         1       308
               44         0         0        40         1         0        40
```

We now have the desired result and are ready to stset our data:

```
. stset t1, failure(died) id(id)

                id:  id
     failure event:  died != 0 & died < .
obs. time interval:  (t1[_n-1], t1]
 exit on or before:  failure

------------------------------------------------------------------------------
      172  total observations
        2  multiple records at same instant                     PROBABLE ERROR
           (t1[_n-1]==t1)
------------------------------------------------------------------------------
      170  observations remaining, representing
      102  subjects
       74  failures in single-failure-per-subject data
    31933  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =      1799
```

Well, something went wrong. Two records were excluded. There are few enough data here that we could just list the dataset and look for the problem, but let's pretend otherwise. We want to find the records that, within patient, are marked as exiting at the same time:

```
. by id: gen problem = t1==t1[_n-1]

. sort id died

. list id if problem

     +----+
     | id |
     |----|
 61. | 38 |
     +----+

. list id transplant wait stime died posttran t1 if id==38

     +--------------------------------------------------------+
     | id   transp~t   wait   stime   died   posttran   t1 |
     |--------------------------------------------------------|
 60. | 38          1      5       5      0          0    5 |
 61. | 38          1      5       5      1          1    5 |
     +--------------------------------------------------------+
```

There is no typographical error in these data—we checked that variables transplant, wait, and stime contain what the original source published. Those variables indicate that patient 38 waited 5 days for a heart transplant, received one on the fifth day, and then died on the fifth day, too.

That makes perfect sense, but not to Stata, which orders events within t as failures, followed by censorings, followed by entries. Reading t1, Stata went for this literal interpretation: patient 38 was censored at time 5 with posttran = 0; then, at time 5, patient 38 died; and then, at time 5, patient 38 reentered the data, but this time with posttran = 1. That made no sense to Stata.

Stata's sequencing of events may surprise you, but trust us, there are good reasons for it, and really, the ordering convention does not matter. To fix this problem, we just have to put a little time between the implied entry at time 5 and the subsequent death:

```
. replace t1 = 5.1 in 61
(1 real change made)

. list id transplant wait stime died posttran t1 if id==38

     +------------------------------------------------------+
     | id   transp~t   wait   stime   died   posttran    t1 |
     |------------------------------------------------------|
 60. | 38          1      5       5      0          0     5 |
 61. | 38          1      5       5      1          1   5.1 |
     +------------------------------------------------------+
```

Now the data make sense both to us and to Stata: until time 5, the patient had posttran $= 0$; then, at time 5, the value of posttran changed to 1; and then, at time 5.1, the patient died.

```
. stset t1, id(id) failure(died)

                id:  id
     failure event:  died != 0 & died < .
obs. time interval:  (t1[_n-1], t1]
 exit on or before:  failure

------------------------------------------------------------------------------
      172  total observations
        0  exclusions
------------------------------------------------------------------------------
      172  observations remaining, representing
      103  subjects
       75  failures in single-failure-per-subject data
  31938.1  total analysis time at risk and under observation
                                                at risk from t =         0
                                     earliest observed entry t =         0
                                          last observed exit t =      1799
```

This dataset is now ready for use with all the other st commands. Here is an illustration:

```
. use http://www.stata-press.com/data/r13/stan3, clear
(Heart transplant data)

. stset, noshow

. stsum, by(posttran)

         |               incidence       no. of    |------ Survival time -----|
posttran | time at risk     rate        subjects        25%       50%       75%
---------+---------------------------------------------------------------------
       0 |         5936   .0050539          103         36       149       340
       1 |      26002.1   .0017306           69         39        96       979
---------+---------------------------------------------------------------------
   total |      31938.1   .0023483          103         36       100       979

. stcox age posttran surgery year

Iteration 0:   log likelihood = -298.31514
Iteration 1:   log likelihood =  -289.7344
Iteration 2:   log likelihood = -289.53498
Iteration 3:   log likelihood = -289.53378
Iteration 4:   log likelihood = -289.53378
Refining estimates:
Iteration 0:   log likelihood = -289.53378

Cox regression -- Breslow method for ties

No. of subjects =          103                 Number of obs    =         172
No. of failures =           75
Time at risk    =      31938.1
                                               LR chi2(4)       =       17.56
Log likelihood  =   -289.53378                 Prob > chi2      =      0.0015

------------------------------------------------------------------------------
          _t | Haz. Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   1.030224   .0143201     2.14   0.032     1.002536    1.058677
    posttran |   .9787243   .3032597    -0.07   0.945     .5332291    1.796416
     surgery |   .3738278    .163204    -2.25   0.024     .1588759       .8796
        year |   .8873107    .059808    -1.77   0.076     .7775022    1.012628
------------------------------------------------------------------------------
```

◁

References

Cleves, M. A. 1999. ssa13: Analysis of multiple failure-time data with Stata. *Stata Technical Bulletin* 49: 30–39. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 338–349. College Station, TX: Stata Press.

Cleves, M. A., W. W. Gould, R. G. Gutierrez, and Y. V. Marchenko. 2010. *An Introduction to Survival Analysis Using Stata*. 3rd ed. College Station, TX: Stata Press.

Crowley, J., and M. Hu. 1977. Covariance analysis of heart transplant survival data. *Journal of the American Statistical Association* 72: 27–36.

Hills, M., and B. L. De Stavola. 2012. *A Short Introduction to Stata for Biostatistics: Updated to Stata 12*. London: Timberlake.

Kalbfleisch, J. D., and R. L. Prentice. 2002. *The Statistical Analysis of Failure Time Data*. 2nd ed. New York: Wiley.

Also see

[ST] **snapspan** — Convert snapshot data to time-span data

[ST] **stdescribe** — Describe survival-time data