serset — Create and manipulate sersets

Description

`serset` creates and manipulates sersets.

`file sersetwrite` writes and `file sersetread` reads sersets into files.

The macro function `:serset` reports information about the current serset.

varlist may contain `strL` variables or `str#` variables. If it does, only the first 244 bytes of each value will be stored in the serset.

Sersets are primarily used by Stata's graphics system. If you are interested in working with multiple datasets in memory, you will want to use Stata's data frames. See [D] **frames**.

Syntax

Create new serset from data in memory

`serset create` *varlist* [*if*] [*in*] [, `omitanymiss omitallmiss`
`omitdupmiss omitnothing sort(`*varlist*`)`]

Create serset of cross medians

`serset create_xmedians` svn_y svn_x [svn_w] [, `bands(#) xmin(#) xmax(#)`
`logx logy`]

Create serset of interpolated points from cubic spline interpolation

`serset create_cspline` svn_y svn_x [, `n(#)`]

Make previously created serset the current serset

`serset` [`set`] $\#_s$

Change order of observations in current serset

`serset sort` [*svn* [*svn* [`...`]]]

Return summary statistics about current serset

`serset summarize` *svn* [, `detail`]

Return in r() information about current serset

`serset`

Load serset into memory

```
serset use [ , clear ]
```

Change ID of current serset

```
serset reset_id #s
```

Eliminate specified sersets from memory

```
serset drop [ numlist | _all ]
```

Eliminate all sersets from memory

```
serset clear
```

Describe existing sersets

```
serset dir
```

The `file` command (see [P] **file**) is also extended to allow

Write serset into file

```
file sersetwrite handle
```

Read serset from file

```
file sersetread handle
```

The following macro functions are also available:

Macro function	Returns from the *current serset*
`: serset id`	ID
`: serset k`	number of variables
`: serset N`	number of observations
`: serset varnum` *svn*	*svnum* of *svn*
`: serset type` *svn*	storage type of *svn*
`: serset format` *svn*	display format of *svn*
`: serset varnames`	list of *svns*
`: serset min` *svn*	minimum of *svn*
`: serset max` *svn*	maximum of *svn*

Macro functions have the syntax

`local` *macname* `: ...`

The *current serset* is the most recently created or the most recently set by the `serset set` command.

In the above syntax diagrams,

$\#_s$ refers to a serset number, $0 \leq \#_s \leq 1{,}999$.

varlist refers to the usual Stata varlist, that is, a list of variables that appear in the current dataset, not the current serset.

svn refers to a variable in a serset. The variable may be referred to by either its name (for example, `mpg` or `l.gnp`) or its number (for example, 1 or 5); which is used makes no difference.

svnum refers to a variable number in a serset.

Options

Options are presented under the following headings:

Options for serset create

`omitanymiss`, `omitallmiss`, `omitdupmiss`, and `omitnothing` specify how observations with missing values are to be treated.

`omitanymiss` is the default. Observations in which any of the numeric variables contain missing are omitted from the serset being created.

`omitallmiss` specifies that only observations in which all the numeric variables contain missing be omitted.

`omitdupmiss` specifies that only duplicate observations in which all the numeric variables contain missing be omitted. Observations omitted will be a function of the sort order of the original data.

`omitnothing` specifies that no observations be omitted (other than those excluded by `if` *exp* and `in` *range*).

`sort(`*varlist*`)` specifies that the serset being created is to be sorted by the specified variables. The result is no different from, after serset creation, using the `serset sort` command, but total execution time is a little faster. The sort order of the data in memory is unaffected by this option.

Options for serset create_xmedians

`bands(#)` specifies the number of divisions along the x scale in which cross medians are to be calculated; the default is `bands(200)`. `bands()` may be specified to be between 3 and 200.

Let m and M specify the minimum and maximum value of x. If the scale is divided into n bands (that is, `bands(`n`)` is specified), the first band is m to $m+(M-m)/n$, the second $m+(M-m)/n$ to $m+2*(M-m)/n$, ..., and the nth $m+(n-1)*(M-m)/n$ to $m+n*(M-m)/n = m+M-m = M$.

`xmin(#)` and `xmax(#)` specify the minimum and maximum values of the x variable to be used in the bands calculation—*m* and *M* in the formulas above. The actual minimum and maximum are used if these options are not specified. Also, if `xmin()` is specified with a number that is greater than the actual minimum, the actual minimum is used, and if `xmax()` is specified with a number that is less than the actual maximum, the actual maximum is used.

`logx` and `logy` specify that cross medians be created using a "log" scale. The exponential of the median of the log of the values is calculated in each band.

Option for serset create_cspline

`n(#)` specifies the number of points to be evaluated between each pair of x values, which are treated as the knots. The default is `n(5)`, and `n()` may be between 1 and 300.

Option for serset summarize

`detail` specifies additional statistics, including skewness, kurtosis, the four smallest and four largest values, and various percentiles. This option is identical to the `detail` option of `summarize`; see [R] **summarize**.

Option for serset use

`clear` permits the serset to be loaded, even if there is a dataset already in memory and even if that dataset has changed since it was last saved.

Remarks and examples

Remarks are presented under the following headings:

Introduction

Sersets are used in implementing Stata's graphics capabilities. When you make a graph, the data for the graph are extracted into a serset and then, at the lowest levels of Stata's graphics implementation, are graphed from there.

Sersets are like datasets: they contain observations on one or more variables. Each serset is assigned a number, and in your program, you use that number when referring to a serset. Thus multiple sersets can reside simultaneously in memory. (Sersets are, in fact, stored in a combination of memory and temporary disk files, so accessing their contents is slower than accessing the data in memory. Sersets, however, are fast enough to keep up with graphics operations.)

serset create

`serset create` creates a new serset from the data in memory. For instance,

```
. serset create mpg weight
```

creates a new serset containing variables `mpg` and `weight`. When using the serset later, you can refer to these variables by their names, `mpg` and `weight`, or by their numbers, `1` and `2`.

`serset create` also returns the following in `r()`:

`r(N)`	the number of observations placed into the serset
`r(k)`	the number of variables placed into the serset
`r(id)`	the number assigned to the serset

`r(N)` and `r(k)` are just for your information; by far the most important returned result is `r(id)`. You will need to use this number in subsequent commands to refer to this serset.

`serset create` also sets the current serset to the one just created. Commands that use sersets always use the current serset. If, in later commands, the current serset is not the one desired, you can set the desired one by using `serset set`, described below.

serset create_xmedians

`serset create_xmedians` creates a new serset based on the currently set serset. The basic syntax is

`serset create_xmedians` svn_y svn_x $[svn_w][$`, ...`$]$

The new serset will contain cross medians. Put that aside. In the `serset create_xmedians` command, you specify two or three variables to be recorded in the current serset. The result is to create a new serset containing two variables (svn_y and svn_x) and a different number of observations. As with `serset create`, the result will also be to store the following in `r()`:

`r(id)`	the number assigned to the serset
`r(k)`	the number of variables in the serset
`r(N)`	the number of observations in the serset

The newly created serset will become the current serset.

In actual use, you might code

```
serset create ‘yvar’ ‘xvar’ ‘zvar’
local base = r(id)
...
serset set ‘base’
serset create_xmedians ‘yvar’ ‘xvar’
local cross = r(id)
...
```

`serset create_xmedians` obtains data from the original serset and calculates the median values of svn_y and the median values of svn_x for bands of svn_x values. The result is a new dataset of n observations (one for each band) containing median y and median x values, where the variables have the same name as the original variables. These results are stored in the newly created serset. If a third variable is specified, svn_w, the medians are calculated with weights.

serset create_cspline

`serset create_cspline` works in the same way as `serset create_xmedians`: it takes one serset and creates another serset from it, leaving the first unchanged. Thus, as with all serset creation commands, returned in `r()` is

`r(id)`	the number assigned to the serset
`r(k)`	the number of variables in the serset
`r(N)`	the number of observations in the serset

and the newly created serset will become the current serset.

`serset create_cspline` performs cubic spline interpolation, and here the new serset will contain the interpolated points. The original serset should contain the knots through which the cubic spline is to pass. `serset create_cspline` also has the `n(#)` option, which specifies how many points are to be interpolated, so the resulting dataset will have $N + (N - 1) * \texttt{n()}$ observations, where N is the number of observations in the original dataset. A typical use of `serset create_cspline` would be

```
serset create ‘yvar’ ‘xvar’
local base = r(id)
...
serset set ‘base’
serset create_xmedians ‘yvar’ ‘xvar’
local cross = r(id)
...
serset set ‘cross’
serset create_cspline ‘yvar’ ‘xvar’
...
```

Here the spline is placed not through the original data but through cross medians of the data.

serset set

`serset set` is used to make a previously created serset the current serset. You may omit the `set`. Typing

```
serset 5
```

is equivalent to typing

```
serset set 5
```

You would never actually know ahead of time the number of a serset that you needed to code. Instead, when you created the serset, you would have recorded the identity of the serset created, say, in a local macro, by typing

```
local id = r(id)
```

and then later, you would make that serset the current serset by coding

```
serset set ‘id’
```

serset sort

`serset sort` changes the order of the observations of the current serset. For instance,

```
serset create mpg weight
local id = r(id)
serset sort weight mpg
```

would place the observations of the serset in ascending order of variable `weight` and, within equal values of `weight`, in ascending order of variable `mpg`.

If no variables are specified after `serset sort`, `serset sort` does nothing. That is not considered an error.

serset summarize

`serset summarize` returns summary statistics about a variable in the current serset. It does not display output or in any way change the current serset.

Returned in `r()` is exactly what the `summarize` command returns in `r()`; see [R] **summarize**.

serset

`serset` typed without arguments produces no output but returns in `r()` information about the current serset:

`r(id)`	the number assigned to the current serset
`r(k)`	the number of variables in the current serset
`r(N)`	the number of observations in the current serset

If no serset is in use, `r(id)` is set to -1, and `r(k)` and `r(N)` are left undefined; no error message is produced.

serset use

`serset use` loads a serset into memory. That is, it copies the current serset into the current data. The serset is left unchanged.

serset reset_id

`serset reset_id` is a rarely used command. Its syntax is

`serset reset_id` $\#_s$

`serset reset_id` changes the ID of the current serset—its number—to the number specified, if that is possible. If not, it produces the error message "series $\#_s$ in use"; r(111).

Either way, the same serset continues to be the current serset (that is, the number of the current serset changes if the command is successful).

serset drop

`serset drop` eliminates (erases) the specified sersets from memory. For instance,

```
serset drop 5
```

would eliminate serset 5, and

```
serset drop 5/9
```

would eliminate sersets 5–9. Using `serset drop` to drop a serset that does not exist is not an error; it does nothing.

Typing `serset drop _all` would drop all existing sersets.

Be careful not to drop sersets that are not yours: Stata's graphics system creates and holds onto sersets frequently, and, if you drop one of its sersets that are in use, the graph on the screen will eventually "fall apart", and Stata will produce error messages (Stata will not crash). The graphics system will itself drop sersets when it is through with them.

The `discard` command also drops all existing sersets. This, however, is safe because `discard` also closes any open graphs.

serset clear

`serset clear` is a synonym for `serset drop _all`.

serset dir

`serset dir` displays a description of all existing sersets.

file sersetwrite and file sersetread

`file sersetwrite` and `file sersetread` are extensions to the `file` command; see [P] **file**. These extensions write and read sersets into files. The files may be opened `text` or `binary`, but, either way, what is written into the file will be in a binary format.

`file sersetwrite` writes the current serset. A code fragment might read

```
serset create ...
local base = r(id)
...
tempname hdl
file open ‘hdl’ using "‘filename’", write ...
...
serset set ‘base’
file sersetwrite ‘hdl’
...
file close ‘hdl’
```

`file sersetread` reads a serset from a file, creating a new serset in memory. `file sersetread` returns in `r(id)` the serset ID of the newly created serset. A code fragment might read

```
tempname hdl
file open ‘hdl’ using "‘filename’", read ...
...
file sersetread ‘hdl’
local new = r(id)
...
file close ‘hdl’
```

See [P] **file** for more information on the `file` command.

Stored results

serset create, serset create_xmedians, serset create_cspline, serset set, and serset store the following in r():

Scalars

r(id)	the serset ID
r(k)	the number of variables in the serset
r(N)	the number of observations in the serset

serset summarize returns in r() the same results as returned by the summarize command.

serset use returns in macro r(varnames) the names of the variables in the newly created dataset.

file sersetread returns in scalar r(id) the serset ID, which is the identification number assigned to the serset.

Also see

[P] **class** — Class programming

[P] **file** — Read and write text and binary files

[D] **frames intro** — Introduction to frames