Sherpa Configuration: Using the State Objects

Sherpa Threads (CIAO 3.4)

Table of Contents

Sherpa Configuration: Using the State Objects

Sherpa Threads

Overview

Last Update: 1 Dec 2006 − reviewed for CIAO 3.4: no changes

Synopsis:

The configuration of many aspects of *Sherpa* is controlled by state objects (a.k.a. configuration variables). This thread introduces the state objects and describes how they can be used to customize various features of *Sherpa*.

Related Links:

- The state objects section of the *Sherpa* ahelp page
- The ahelp pages for individual state objects: `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, `sherpa.resplot`, `sherpa.multiplot`, `sherpa.output`, `sherpa.regproj`, `sherpa.regunc`, `sherpa.intproj`, `sherpa.intunc`, `sherpa.proj`, `sherpa.cov`, `sherpa.unc`
- For details on configuring plots:
  - Step−by−Step guide to changing the look of Sherpa plots
  - Advanced customization of Sherpa plots
- For details on configuring confidence−level calculations: Step−by−Step Guide to Estimating Errors and Confidence Levels

Proceed to the HTML or hardcopy (PDF: A4 | letter) version of the thread.

State Object Basics

The *Sherpa* state object (a.k.a. configuration variable) is a S−Lang variable that is initialized whenever one starts a *Sherpa* session or loads the *Sherpa* S−Lang module. The contents of the state object can be displayed with the `print` function:

```
sherpa> print(sherpa)
plot                =   sherpa_Plot_State
dataplot            =   sherpa_Plot_State
fitplot             =   sherpa_FitPlot_State
resplot             =   sherpa_Plot_State
multiplot           =   sherpa_Draw_State
output              =   sherpa_Output_State
regproj             =   sherpa_VisParEst_State
regunc              =   sherpa_VisParEst_State
intproj             =   sherpa_VisParEst_State
intunc              =   sherpa_VisParEst_State
```

```
proj             =  sherpa_Proj_State
cov              =  sherpa_Cov_State
unc              =  sherpa_Unc_State
con_levs         =  NULL
modeloverride    =  0
multiback        =  0
deleteframes     =  1
clobber          =  0
```

The column to the left of the equals signs (=) contains the names of the state object's fields, and the column to the right contains the fields' values. Some of the state object's fields (e.g. `multiback`) contain simple, atomic values, such as integers or strings; the values of these fields appear in the output of `print(sherpa)`. Others fields (i.e. `plot` and those whose values are of the form `sherpa_..._State`) are state objects themselves, which contain fields of their own. To display the contents of these state objects, use the syntax `print(sherpa.<name>)`, as shown below for `sherpa.plot`.

You can select a specific field of a state object using the dot (`.`) operator:

```
sherpa> print(sherpa.modeloverride)
0
sherpa> print(sherpa.plot)
x_errorbars      =  0
y_errorbars      =  0
errs_style       =  bar
errs_type        =  both
...
sherpa> print(sherpa.plot.errs_style)
bar
```

Each atomic field is a S−Lang variable and can be set using the syntax "`<name> = <value>`". For example, the following commands set `sherpa.modeloverride` to 1 and `sherpa.plot.errs_style` to "standard":

```
sherpa> sherpa.modeloverride = 1
sherpa> sherpa.plot.errs_style = "standard"
```

Note that string values *must* be enclosed in double quotes. The reason is that S−Lang interprets `"standard"` (with quotes) as a literal string, whereas `standard` (without quotes) is a variable name.

The sections that follow introduce the state object fields and explain how they affect various features of *Sherpa*. For more detailed information, see the "Related Links" section of the Overview.

Note that the state object concept is not unique to *Sherpa*. For example, both Varmm and *ChIPS* have state objects of their own. Since *Sherpa* uses Varmm and *ChIPS*, these state objects are also available within a *Sherpa* session.

Changing the Appearance of Plots

Five *Sherpa* state objects control the appearance of plots. The following table lists these state objects and the plot types that they control:

State object	Plot types
`sherpa.plot`	The output of `LPLOT`, `OPLOT`, `CPLOT`, and `SPLOT` for all plots except data, fit, and residual plots
`sherpa.dataplot`	Data plots (`LPLOT DATA` and `LPLOT BACK`)
`sherpa.fitplot`	Fit plots (`LPLOT FIT` and `LPLOT BFIT`)

sherpa.resplot	Residual plots (LPLOT RESIDUALS, LPLOT RATIO, LPLOT BRESIDUALS, and LPLOT BRATIO)
sherpa.multiplot	*All* plots

The ahelp files for the individual state objects describe their fields and the plotting options that they control. For more information on configuring plots via the state objects, see the thread "Step−by−Step guide to changing the look of Sherpa plots". For information on how to use the `setplot.sl` script to simplify plot configuration, see the thread "Changing the look of Sherpa plots using setplot.sl".

Choosing Linear or Logarithmic Scales

One commonly−used plot configuration option is to change from linear to logarithmic axes (or vice versa). The state objects `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, and `sherpa.resplot` contain the fields `x_log` and `y_log`, which control the axis scales. For example, to produce data plots with a linear x axis and logarithmic y axis, you would set these fields as follows:

```
sherpa> sherpa.dataplot.x_log = 0
sherpa> sherpa.dataplot.y_log = 1
```

However, *Sherpa* also provides convenience functions that allow you to change the axis scales for *all* plots at once. The functions `set_xlog`, `set_ylog`, and `set_log` set logarithmic scales for the x axis, y axis, and both axes, respectively, for all plot types. Similarly, `set_xlin`, `set_ylin`, and `set_lin` set the scales for the corresponding axes to linear. Hence, the changes made above for `sherpa.dataplot` could be made for all plots at once as follows:

```
sherpa> set_xlin
sherpa> set_ylog
```

Controlling Display of Error Bars

Sherpa provides similar functions for controlling the drawing of error bars. `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, and `sherpa.resplot` contain the fields `x_errorbars` and `y_errorbars`, which specify whether error bars should be drawn. To turn error bars on for all plot types, use the functions `set_xerron`, `set_yerron`, and `set_erron`. To turn error bars off for all plot types, use `set_xerroff`, `set_yerroff`, and `set_erroff`.

Advanced Customization with Function Hooks

All of the plotting state objects contain the fields `prefunc` and `postfunc`. These allow the user to define a S−Lang function that is always executed before or after a plot is made, respectively, providing virtually unlimited control over the appearance of plots. For more information on using these fields, see the `sherpa−plot−hooks` ahelp file and the thread "Advanced customization of Sherpa plots".

Modifying Existing Plots

Any changes made to the plotting state objects apply only to plots created after the changes are made; they do *not* apply to existing plots. Hence, running `set_xlog` after "`LPLOT DATA`" will not make the scale of the plot's x axis logarithmic.

To change the appearance of an existing plot (without reissuing the `LPLOT` command), you must use *ChIPS* commands. For example, to make the x axis of an existing plot logarithmic, use the command "`LOG X`" followed by `REDRAW`.

Configuring Confidence−Level Calculations

Seven *Sherpa* state objects control the configuration of confidence−level calculations. The following table lists these state objects and the corresponding *Sherpa* command that they configure:

State object	Corresponding command
`sherpa.regproj`	`REGION−PROJECTION`
`sherpa.regunc`	`REGION−UNCERTAINTY`
`sherpa.intproj`	`INTERVAL−PROJECTION`
`sherpa.intunc`	`INTERVAL−UNCERTAINTY`
`sherpa.proj`	`PROJECTION`
`sherpa.cov`	`COVARIANCE`
`sherpa.unc`	`UNCERTAINTY`

The ahelp files for the individual state objects describe them in detail. For an introduction to using these state objects, see the thread "Step−by−Step Guide to Estimating Errors and Confidence Levels". For information on how to use the `paramest.sl` script to simplify confidence−level calculations, see the thread "Estimating Errors and Confidence Levels".

Displaying the Current Settings

Sherpa provides a number of functions for displaying the current and default settings of the confidence−level state objects. For example, to show the current and default settings for `sherpa.proj`, use the `list proj` function:

```
sherpa> list proj
Parameter       Current         Default                        Description
---------------------------------------------------------------------------
fast                  1               1    Switch to LM/simplex: 0(n)/1(y)
sigma                 1               1                    Number of sigma
```

Restoring the Default Settings

Sherpa also provides functions for restoring the default settings of the confidence−level state objects. For example, to restore the default settings of `sherpa.proj`, use the `restore proj` function:

```
sherpa> restore proj
```

Other Configuration Options

Along with plots and confidence calculations, the *Sherpa* state object controls several other aspects of *Sherpa*'s behavior.

Floating−Point Output

The `sherpa.output` state object controls the appearance of floating−point numbers that are printed to the screen (i.e. standard output). For example, you can tell *Sherpa* to display all floating−point numbers in scientific notation by setting the `scientific` field to 1:

```
sherpa> sherpa.output.scientific = 1
```

Controlling Contour Levels

The CPLOT command generates contour plots of 2−D data, models, etc. By default, *Sherpa* automatically calculates values for the contour levels. However, the user can specify contour levels by setting `sherpa.con_levs` to an array of values:

```
sherpa> # Default is sherpa.con_levs = NULL (sherpa calculates levels)
sherpa> print(sherpa.con_levs)
NULL
sherpa> # Set levels to 0.5, 1.5, 3
sherpa> sherpa.con_levs = [ 0.5, 1.5, 3 ]
```

Note that `sherpa.con_levs` affects only CPLOT. It has no influence on the plots generated by REGION−PROJECTION or REGION−UNCERTAINTY.

Model Overriding

The state object field `sherpa.modeloverride` affects how *Sherpa* handles the creation of models. If it is set to 1, then a model can be redefined without first being erased with the ERASE command:

```
sherpa> print(sherpa.modeloverride)
0
sherpa> powlaw1d[m]
sherpa> gauss[m]
Error: model component name m is already in use.
       If the last command was to use m in a model stack,
       that stack has been deleted.
sherpa> sherpa.modeloverride = 1
sherpa> gauss[m]
sherpa> show m
gauss1d[m]  (integrate: on)
...
```

Multiple Backgrounds Per Dataset

By default, *Sherpa* allows only one background data file per dataset. However, if `sherpa.multiback` is set to 1, multiple background files are permitted:

```
sherpa> sherpa.multiback = 1
```

See the BACK command for more information on using multiple background files.

Deleting ds9 Frames

The IMAGE command sends 2−D images to ds9 for display. By default, *Sherpa* deletes all existing ds9 frames before sending data to a newly−created frame. However, if `sherpa.deleteframes` is set to 0, then existing frames will *not* be deleted (the data will still go to a new frame):

```
sherpa> # Default is to delete existing frames (sherpa.deleteframes = 1)
sherpa> print(sherpa.deleteframes)
1
sherpa> # Disable frame deletion
sherpa> sherpa.deleteframes = 0
```

File Overwriting

By default, the `WRITE` command will *not* overwrite existing files. The user can change this behavior by setting `sherpa.clobber` to 1:

```
sherpa> sherpa.clobber = 1
```

Saving and Restoring State Object Settings

The `save_state` function can be used to save the current state−object settings so that they can be used in another *Sherpa* session.

If called with no arguments, `save_state` will write out the contents of all fields of all *Sherpa* state objects to the file `$HOME/.sherpa−state−rc`:

```
sherpa> save_state
sherpa> $ more $HOME/.sherpa−state−rc
% Sherpa state for ciaouser, Wed Jul 30 17:45:13 2006

sherpa.plot.x_errorbars = 0;
sherpa.plot.y_errorbars = 0;
sherpa.plot.errs_style = "bar";
sherpa.plot.errs_type = "both";
...
```

This file will be over−written without warning, so you should not make changes to it manually. When *Sherpa* starts, it will automatically load in the settings from this file, so any customizations your have made to the state object will be restored.

If called with a filename as its argument, `save_state` will write out the settings to the specified file (instead of `$HOME/.sherpa−state−rc`):

```
sherpa> save_state("my_settings.shp")
sherpa> $ more my_settings.shp
% Sherpa state for ciaouser, Wed Jul 30 17:45:13 2006

sherpa.plot.x_errorbars = 0;
sherpa.plot.y_errorbars = 0;
...
```

This file can then be read into a *Sherpa* session via the `USE` command. This can be useful, for example, if you want to set up different plot styles for use in different situations.

Creating Aliases for State Objects

The names of the *Sherpa* state objects and their fields are verbose. This can be a benefit, as longer names are more descriptive and give one a better sense of what particular objects and fields control. However, longer names also require more typing, which can be a nuisance when making frequent changes. To alleviate this problem, one may define aliases for state objects. For example, you can make `dp` an alias for `sherpa.dataplot` as follows:

```
sherpa> dp = sherpa.dataplot
```

This command creates a S−Lang variable named `dp` that contains a reference to the `sherpa.dataplot` state object, allowing one to use `dp` as an alias for the full name:

```
sherpa> print(sherpa.dataplot.x_log)
0
sherpa> dp.x_log = 1
sherpa> print(sherpa.dataplot.x_log)
1
```

If you find aliases helpful, you can add lines to create them to your *Sherpa* resource file. This will make them available during every *Sherpa* session.

Note that you can create aliases for state objects but *not* for atomic fields within state objects. For example, you can create an alias for `sherpa` or `sherpa.multiplot` but not for `sherpa.modeloverride`.

History

14 Jan 2005	reviewed for CIAO 3.2: no changes
21 Dec 2005	reviewed for CIAO 3.3: no changes
01 Dec 2006	reviewed for CIAO 3.4: no changes

URL: http://cxc.harvard.edu/sherpa/threads/state_objects/ Last modified: 1 Dec 2006