The *Sherpa* Reference Manual

CIAO Software Release v3.4

December 2006

http://cxc.harvard.edu/sherpa/

Contents

List of Tables

Chapter 1

Sherpa Commands

This section contains all *Sherpa* native commands which can be typed at the prompt or included in the input ASCII file. *Sherpa* also allows the access to the internal information (e.g. about data, statistics, errors etc.) through many S-lang access functions. For a list of the available functions see Section 8.

1.1 Summary of Commands

Sherpa is the generalized fitting engine of *CIAO*. *Sherpa* enables the user to fit models to data, particularly but not exclusively, to data that is being returned by *NASA's Chandra X-ray Observatory*. *Sherpa* features syntax that allows the user to construct complex models from simple definitions and to link parameters algebraically.

Below is a list of all *Sherpa* commands, along with a brief description. *Sherpa* also understands the S-Lang programming language, as described in `AHELP SLANG-CIAO` and `AHELP SLANG-SHERPA`.

`Command`	**Description**
`$`	Must precede any *UNIX* command.
`# | %`	Denotes a comment.
`ANALYSIS`	Specifies whether to analyze a given dataset in energy, wavelength, or channel space.
`BACK`	Inputs the contents of one or more background data files.
`BACKERRORS | BERRORS`	Defines an expression to be used for the background errors.
`BACKGROUND | BG`	Defines a model expression to be used for the background.
`BDCOUNTS`	Calculates the sum of observed counts data for background datasets.
`BEFLUX`	Calculates the unconvolved energy flux for background datasets.
`BGROUP`	Applies grouping scheme to background data.
`BMCOUNTS`	Calculates the sum of convolved model amplitudes for background datasets.
`BSTATERRORS`	Defines the statistical errors for background data.
`BSYSERRORS`	Defines systematic errors for background data.
`BUNGROUP`	Undo grouping applied to background data.

Command	**Description**
`BYE | EXIT | QUIT`	Terminates the *Sherpa* program.
`CAST`	Casts a dataset to single or double precision.
`CLOSE`	Closes the image display window.
`COORD`	Specifies the coordinate system for use in fits to 2-D images.
`COVARIANCE`	Computes covariance matrices and confidence intervals for thawed parameters.
`CPLOT`	Causes the specified 2-D data to be displayed, with a contour plot.
`CREATE`	Establishes a model component and its parameters.
`DATA`	Inputs the contents of one or more source data files.
`DATASPACE`	Sets a value range on which a source model may be evaluated, when no dataset has been input.
`DCOUNTS`	Calculates the sum of observed counts data for source datasets.
`ECHO`	Turns on/off reporting of user keystrokes.
`EFLUX`	Calculates the unconvolved energy flux for source datasets.
`EQWIDTH`	Computes the equivalent width of an emission or absorption line.
`ERASE`	Erases user inputs and settings, plotting objects, and/or model components.
`ERRORS`	Defines an expression to be used for the source errors.
`EXIT | BYE | QUIT`	Terminates the *Sherpa* program.
`EXPMAP`	Defines the exposure map model to be used.
`FAKEIT`	Creates a simulated 1-D dataset.
`FEFFILE`	Inputs a Fits Embedded Function (FEF) file.
`FEFPLOT`	Plots an instrument response stored in Fits Embedded Function (FEF) format.
`FIT | RUN`	Initiates fitting.
`FLUX`	Calculates the model or counts flux for 1-D data.
`FREEZE`	Prohibits a model parameter from varying.
`FTEST`	Computes significance using the F test.
`GETX`	Assigns x-axis values taken from a plot to model parameters.
`GETY`	Assigns y-axis values taken from a plot to model parameters.
`GOODNESS`	Reports information about the χ^2 goodness-of-fit.
`GROUP`	Applies grouping scheme to source data.
`GUESS`	Estimates initial parameter values and ranges, based on input data.
`IGNORE`	Specifies data to be excluded from analyses.
`IMAGE`	Causes the specified 2-D data to be displayed.
`INSTRUMENT | RESPONSE`	Defines the instrument model to be used.
`INTEGRATE`	Controls the integration of model components.
`INTERVAL-PROJECTION | INT-PROJ`	Plots the fit statistic as a function of parameter value, using the `PROJECTION` algorithm.
`INTERVAL-UNCERTAINTY | INT-UNC`	Plots the fit statistic as a function of parameter value, using the `UNCERTAINTY` algorithm.
`JOURNAL`	Turns on/off writing of all commands to a file.
`LPLOT`	Causes the specified 1-D data to be displayed.
`MCOUNTS`	Calculates the sum of convolved model amplitudes for source datasets.
`METHOD | SEARCHMETHOD`	Specifies the optimization method.
`MLR`	Computes significance using the Maximum Likelihood Ratio test.
`NOTICE`	Specifies data to be included in analyses.
`OPEN`	Opens the specified data display window.

Command	**Description**
OPLOT	Causes multiple data curves to be displayed in the same drawing area.
PARAMPROMPT	Turns on/off prompting for model parameter values.
PLOTX	Sets the unit type for the x-axis of the plot.
PLOTY	Sets the unit type for the y-axis of the plot.
PRECISION	Controls the precision of numerical values displayed with SHOW.
PROJECTION	Estimates confidence intervals for thawed parameters.
PROMPT	Changes the *Sherpa* prompt.
QUIT | BYE | EXIT	Terminates the *Sherpa* program.
READ	Inputs the contents of one or more files.
RECORD	Toggles on/off writing of fit parameter values and statistics.
REGION-PROJECTION | REG-PROJ	Creates a contour plot of confidence regions, computed using the PROJECTION algorithm.
REGION-UNCERTAINTY | REG-UNC	Creates a contour plot of confidence regions, computed using the UNCERTAINTY algorithm.
RENAME	Changes the name that has been given to a model component.
RESET	Restores settings and/or parameter values.
RESPONSE | INSTRUMENT	Defines the instrument model to be used.
RUN | FIT	Initiates fitting.
SAVE	Causes settings, definitions, filters, and/or parameter values to be written to an ASCII file.
SEARCHMETHOD | METHOD	Specifies the optimization method to be used.
SET	Sets options for plots generated with LPLOT.
SETBACK	Sets time and backscal attributes for background datasets.
SETDATA	Sets time and backscal attributes for datasets.
SHOW	Reports current status.
SOURCE | SRC	Defines the source model expression to be used for fitting a dataset.
SPLOT	Causes the specified 2-D data to be displayed, with a surface plot.
STATISTIC	Specifies the fitting statistic.
SUBTRACT	Performs background subtraction.
STATERRORS	Defines the statistical errors for source data.
SYSERRORS	Defines systematic errors for source data.
THAW	Allows a model parameter to vary.
TRUNCATE	Resets negative model amplitudes to zero.
UNCERTAINTY	Estimates confidence intervals for thawed parameters.
UNGROUP	Undo grouping applied to source data.
UNLINK	Removes a link between model parameters.
UNSUBTRACT	Performs an undo of background subtraction.
USE	Calls and executes a *Sherpa* script.
VERSION	Reports the *Sherpa* version that is in use.
WCS	Sets the WCS for use as the 2D coordinate system.
WEIGHT	Assigns a weight value to data.
WRITE	Causes the specified information to be written to a file.
XSPEC ABUNDAN	Performs the *XSPEC*command **abund**, which sets abundancies.
XSPEC XSECT	Performs the *XSPEC*command **xsect**, which sets the photoelectric cross-section.

Table 1.1: Summary of Commands

1.2 Description Conventions

In this Chapter, we describe all *Sherpa* commands, in alphabetical order by command name. These descriptions are meant to serve as a quick reference for *Sherpa* users. Beginning *Sherpa* users may wish to first review *Sherpa* Threads, available from

`http://cxc.harvard.edu/sherpa/`

before utilizing this reference.

Each command section includes a description of the command purpose and syntax, along with any other relevant information and examples. Note that each section's set of examples should be run from a newly begun *Sherpa* session. When there are multiple examples within a section, they should be run consecutively.

Information about commands is also available within *Sherpa* with the command `AHELP`.

The following conventions are herein used for describing the syntax:

- `sherpa>` is the default *Sherpa* command-prompt.
- <`arg`> is an argument to a command.
- `#` is an integer only argument.
- Brackets, [ ], indicate an optional argument, *unless otherwise noted.*
- Pipe, `|`, means 'or'.
- {<`arg_a`> `...|` <`arg_b`>`...`} indicates a choice between different command argument sequences.
- {<`sherpa_command_a`> `|` <`sherpa_command_b`>} indicates a choice between equivalent commands.
- Backslash, \, is used to continue a lengthy command onto a second line.
- Non-variable command elements are capitalized in the following descriptions. However, *Sherpa* commands are *not case sensitive* and so may be issued in either upper or lower case.

1.3 Running *Sherpa*

Once *CIAO* has been installed, *Sherpa* may be launched by typing `sherpa` on the command line. Or, *Sherpa* may be launched using a command-line option:

```
SHERPA, version 2.2
Copyright (C) 1999-2001, Smithsonian Astrophysical Observatory

Usage: sherpa [options] [FILE]

Options:
```

```
  --slscript  SCRIPT      Run the given S-Lang script before launching sherpa.
                          This script may contain any valid S-Lang construct,
                          including sherpa_eval() and varmm I/O statements.
  -h, -help, --help       Display this message
  --batch                 Don't show copyright; exit after loading FILE

FILE is optional, and may contain only sherpa commands or simple (one line)
S-Lang statements.
```

The user may create customized *Sherpa* defaults, by creating a file in their home directory named `.sherparc` that contains *Sherpa* commands. For example, the following `.sherparc` file will cause prompting for model parameter values to be turned `OFF`, and will change the default optimization method from `POWELL` to `LEVENBERG-MARQUARDT`:

```
unix% more ~jsmith/.sherparc
PARAMPROMPT OFF
METHOD LEVENBERG-MARQUARDT
```

1.4 $

Must precede any Unix command issued within *Sherpa* or *ChIPS*.

```
sherpa> $<arg>
chips> $<arg>
```

where <`arg`> is a Unix command. Alternatively, ! can be used in place of $.

However, the Unix command `cd` is an exception to this syntax, and must be issued within *Sherpa* or *ChIPS* without a preceding `$` character.

The Unix commands `ls` and `pwd` are also exceptions to this syntax, and may or may not be issued within *Sherpa* or *ChIPS* without a preceding `$` character.

Examples:

1. Issue various Unix commands within *Sherpa*:

```
sherpa> cd /disks/a/mydata
sherpa> ls
sherpa> pwd
/disks/a/mydata
```

2. Issue various Unix commands within *Sherpa*:

```
sherpa> $more data.dat
0.5 1.80766
1.5 2.21929
2.5 2.64117
3.5 3.10638
4.5 3.51711
5.5 3.74749
```

3. Use a Unix command to clear the screen:

```
sherpa> $clear
```

This Unix command will clear the screen.

1.5 # — %

Denotes a comment. The symbol % also denotes a comment.

```
sherpa> {# | %} <arg>

chips> {# | %} <arg>

where <arg> is a comment.
```

Examples:

1. Issue a comment:

```
sherpa> # My comment
sherpa> POLY[modela]
```

2.
```
chips> # My comment
chips> CURVE data/example.dat X 1 Y 2
```

1.6 ANALYSIS

Specifies whether to analyze datasets in energy, wavelength, or channel space.

```
sherpa> ANALYSIS [<dataset range> | ALLSETS] \
[{ ENERGY | WAVELENGTH | CHANNELS | BINS }]
```

where <`dataset range`> = `#, or more generally #:#,#:#,...`, such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default for `ANALYSIS` is all datasets (`ALLSETS`).

The WAVELENGTH argument may be shortened to WAVE.

The analysis setting for each specified dataset is shown if the final argument is omitted, *e.g.* "ANALYSIS ALLSETS".

The analysis setting is automatically determined when data are read in. This setting can be altered with the `ANALYSIS` command. In general, the analysis setting for non-PHA data is `CHANNELS` (synonymous with `BINS`), while for general PHA files it is `ENERGY`. When the columns `BIN_LO` and `BIN_HI` are detected in a Type II PHA file (such as can be the case for files containing *Chandra* grating data), these are assumed to contain wavelength information and the analysis is set to `WAVELENGTH`.

PHA datasets for which there is an `INSTRUMENT` model set may be analyzed in either `ENERGY` or `WAVE` space. Setting `ANALYSIS` to `WAVE` or `ENERGY` has no effect until an ARF and/or an RMF are read in and an `INSTRUMENT` (Section 1.46) is specified.

If the `ANALYSIS` setting is changed after the model components and their parameter values and ranges have been set, issue the `GUESS` command (Section 1.43) to reset the initial values and ranges.

Some models, such as certain XSPEC models, expect that the x-values will always be energy bins. When the analysis setting is using non-energy bins (*e.g.*, `WAVE`) and an XSPEC model is defined, *Sherpa* converts the bins to energy before sending them to the XSPEC model. After the XSPEC model finishes, *Sherpa* converts back to the original units. *Sherpa* also scales the model values appropriately (*e.g.*, if counts/keV came out of the XSPEC model, and *Sherpa* is working with wavelength bins, then *Sherpa* scales the output of the XSPEC model to counts/Angstrom).

Examples:

1. Read in *Chandra* grating data and a grating RMF. Analyze the O VIII Lyman α line at 18.97 Å in energy space, in the dataset (# 9) containing MEG -1 order counts:

```
sherpa> DATA data_pha2.fits
The inferred file type is PHA Type II.  If this is not what you want, please
specify the type explicitly in the data command.
Warning: could not find SYS_ERR column
WARNING: statistical errors specified in the PHA file.
         These are currently IGNORED.  To use them, type:
         READ ERRORS "<filename>[cols CHANNEL,STAT_ERR]" fitsbin
WARNING: backgrounds UP and DOWN are being read from this file,
         and are being combined into a single background dataset.
WARNING: multiple datasets have been input.
         The next available dataset number is 13.
sherpa> PARAMPROMPT OFF
```

```
sherpa> RSP[a]
sherpa> a.rmf = meg1.rmf
sherpa> a.arf = meg1.arf
sherpa> INSTRUMENT 9 = a
sherpa> IGNORE 9 ALL
sherpa> NOTICE 9 WAVE 18.8:19.2
sherpa> SOURCE 9 = CONST[co] + GAUSS[g]
sherpa> FIT 9
 LVMQT: V2.0
 LVMQT: initial statistic value = 42730.9
 LVMQT: final statistic value = 21.8345 at iteration 74
            co.c0  0.000378496
             g.fwhm  0.00194331
             g.pos  18.9812
             g.ampl  1.53444
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed 1.9433e-03 6.8361e-04     6.8361
 2    pos thawed    18.9812    18.7975    19.1975
 3   ampl thawed     1.5344 1.3713e-03    13.7131
sherpa> ANALYSIS 9 ENERGY
sherpa> GUESS SOURCE 9
sherpa> FIT 9
 LVMQT: V2.0
 LVMQT: initial statistic value = 43170.1
 LVMQT: final statistic value = 21.7635 at iteration 66
            co.c0  0.0112294
             g.fwhm  0.000107187
             g.pos  0.653184
             g.ampl  27.7522
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed 1.0719e-04 2.3488e-05     0.2349
 2    pos thawed     0.6532     0.6457     0.6594
 3   ampl thawed    27.7522 3.9868e-02   398.6771
```

1.7 BACK

Inputs the contents of one or more background data files.

```
sherpa> BACK [# [ID]] <filespec> [,[# [ID]] <filespec>,...]
```

where `#` specifies the number of the dataset to be associated with this background data file (default dataset number is 1). The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

Alternative means of reading in background datasets involve using the `load` functions of the *Sherpa*/S-Lang module (*e.g.*, `load_bpha`).

The help file for the `READ` (Section 1.64) command explains the <filespec> definition and has a listing of allowed file types; see also the related commands `BACKGROUND` (Section 1.9), `BACKERRORS` (Section 1.8), and `SETBACK` (Section 1.74).

A few things to note:

- If the source data are read in after the background data are read in, the background data are deleted. Always read in the source data first!
- To perform background subtraction in *Sherpa*, the command `SUBTRACT`, Section 1.80, must be issued; this is in contrast to *XSPEC*, which performs background subtraction automatically.
- Input source and background datasets do not need to be the same size *unless* you wish to subtract the background. The `SUBTRACT` command may only be used when the input source and background datasets are the same size. (More specifically, for the case of PHA datasets, the source and background datasets must have the same size when ungrouped; if the input source dataset is grouped and the input background dataset is ungrouped, *Sherpa* will automatically group the background data before subtraction.)
- In CIAO 3.1 the definition of INSTRUMENT BACK is required for both filtering and fitting of PHA data if either background file or background models have been defined. INSTRUMENT BACK is set automatically when the PHA source file is input to *Sherpa*, however it is deleted if the NEW background file is input for a given data set, thus the new INSTRUMENT BACK has to defined on the command line before filtering and fitting the data with the new background file.

1.7.1 Using Data Model Filters

This command is able to take any Data Model virtual file specification (see "ahelp dmsyntax"). If you can do

```
unix% dmcopy "infile.fits[spec 1][spec 2]" outfile.fits
```

you can also do

```
sherpa> back "infile.fits[spec 1][spec 2]"
```

This is especially useful when working with very large files. For example:

```
sherpa> back "evt.fits[bin sky=4][opt mem=100]"
```

bins the event file by a factor of four and allocates additional memory. A similar command (omitting the binning factor) can be used to read in an image.

Examples:

1. Input 2-D FITS image data and background files; subtract and unsubtract background data:

```
sherpa> DATA 3 example_img.fits FITS
sherpa> BACK 3 example_img_bkg.fits FITS
sherpa> SUBTRACT 3
sherpa> UNSUBTRACT 3
```

   The first command, `DATA` 3 example_img.fits FITS, reads the FITS image example_img.fits, as dataset number 3. The second command reads the background FITS image for this dataset. The third command, `SUBTRACT 3`, performs the background subtraction for dataset number 3. The final command, `UNSUBTRACT 3`, restores dataset number 3 to its original unsubtracted state.

2. Input IMH image data and background files, then subtract the background:

```
sherpa> DATA example.imh
sherpa> BACK example_bkg.imh
sherpa> SUBTRACT
```

   The first two commands read in the image and background image, respectively, for dataset 1. `SUBTRACT` subtracts the background from the data.

1.8 BACKERRORS — BERRORS

Defines an expression to be used to specify the *statistical* errors for background data. The commands `BACKERRORS` and `BSTATERRORS` are equivalent.

```
sherpa> BERRORS [<dataset range> | ALLSETS] [ID] = <errorExpr>
```

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1. The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The error expression, <errorExpr>, may be composed of one or more (algebraically-combined) of the following elements:

Component:	Description:
`BACK`	An input background dataset
numericals	Numerical values
operators	+ - * / ( )

A few things to note:

- Background error expressions cannot be defined without first reading in a background dataset.
- The background errors can be specified in an input file; see `READ BERRORS` (Section 1.64) for more information.
- If the background errors are not defined using `BERRORS` or input using `READ BERRORS`, then they are computed as defined for the current statistic; see the documentation on the `STATISTIC` (Section 1.79) command.

See the related command `BSYSERRORS` (Section 1.15).

The background errors are accessible to the *Sherpa*/S-Lang module user via the functions **`get_berrors`** and **`set_berrors`**.

Examples:

1. Define an expression to be used for the background errors. They are set to 1.5 in each bin.

   ```
   sherpa> DATA data.dat
   sherpa> BACK back.dat
   sherpa> BERRORS = 1.5
   ```

2. Define an expression to be used for the background errors. They are set to 0.5 times the background datum in each bin.

   ```
   sherpa> BERRORS = 0.5*BACK
   ```

1.9 BACKGROUND — BG

Defines a model expression to be used for the background. The command BG is an abbreviated equivalent.

```
sherpa> BACKGROUND [<dataset range> | ALLSETS [ID]] = <modelExpr>
```

<dataset range> = `#` (or more generally `#:#,#:#`, *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1. The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The model expression, <`modelExpr`>, is an algebraic combination of one or more of the following elements:

```
{<sherpa_modelname> | <sherpa_modelname>[modelname] |
 <modelname> | <model_stack> | <nested_model>}
```

along with numerical values. The following operators are recognized: + - * / ( ) { }. See the `CREATE` (Section 1.23) command for further information.

Note that:

- The documentation on *Sherpa Models* (Section 5) contains a summary list, and descriptions, of the models that are available within *Sherpa*, which include models from *XSPEC*, v. 11.3.
- By default, if the model expression includes a model component that has not previously been established, *Sherpa* will prompt for the initial parameter values for that model component. This prompting can be turned off using the `PARAMPROMPT OFF` (Section 1.58) command.
- In CIAO 3.1 the definition of INSTRUMENT BACK is required for both filtering and fitting of PHA data if either background file or background models have been defined. INSTRUMENT BACK is set automatically when the PHA source file is input to *Sherpa*, however it is deleted if the NEW background file is input for a given data set, thus the new INSTRUMENT BACK has to defined on the command line before filtering and fitting the data with the new background file.

To reset a background model stack, issue the command:

```
sherpa> BACKGROUND [<dataset range> | ALLSETS] =
```

How the background model stack is used depends upon whether or not the the source data have been *background-subtracted*:

- If the source data are background-subtracted, then the background model stack is applied only to the fit of the background data themselves. The resulting statistic is added to that from the source model fit; changing a background model parameter has no effect on the source fit.
- If the source data are not background-subtracted, then the background model stack is evaluated twice, once on a grid appropriate for the background data and a second time on a grid appropriate to the source data. To the latter array of amplitudes is added the evaluated source model amplitudes. Thus changing a background model parameter affects the fit statistic in both the background fit and the source fit!

Note on Model Normalization. Because the background is, by definition, an "extended object," the normalization of a best-fit background model will be affected by the size of the background extraction region, which is proportional to the area of the sky from which the photons came. In particular, if the areas of the source and background extraction regions differ, then the normalization may not be easily interpretable: which region does it correspond to? In CIAO 3.0, the rules are the following:

- If source and background data have both been input, the best-fit background normalization corresponds to the source extraction region, with one exception noted below.
- If background data only have been input, the best-fit background normalization corresponds to the background extraction region.

- If source and background data have both been input, but the ratio of extraction "areas" differs as a function of energy (as it can for, *e.g.*, *XMM* grating data), then the normalization corresponds to the pixel area equivalent to a `BACKSCAL` of one.

Examples:

1. Define a model to be used for the background and set background model parameter values:

```
sherpa> DATA 2 data.dat
sherpa> BACKGROUND 2 = GAUSS
GAUSS.fwhm parameter value [10]
GAUSS.pos parameter value [0] 3
GAUSS.ampl parameter value [1] 2:1:10
```

   This command defines the *Sherpa* model `GAUSS` as the background model for dataset number 2. The user accepted the given initial guessed value for the parameter `fwhm` (using the <RETURN> key), entered a value of 3 for parameter `pos`, and entered a value of 2 (with min:max range of 1:10) for parameter `ampl`.

2. Define a model to be used for the background and set the parameter values:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> DATA data.dat
sherpa> POISSON[bkgA]
sherpa> BACKGROUND = bkgA
```

   In the third command, the name `bkgA` is given to the *Sherpa* model component `POISSON`. The final command defines this model as the model to be used for the background.

3. Create a background model expression:

```
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> BACKGROUND = (POW[modelc])/2
modelc.gamma parameter value [0]
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
```

   This command assigns the model expression `(POW[modelc])/2`, to the background model for dataset number 1. In this example, the user accepted the given initial values for all of the parameters via parameter prompting.

   See the `SOURCE` command documentation for more (analogous) examples.

1.10 BDCOUNTS

Calculates the sum of observed counts data for background datasets.

See the `DCOUNTS` command (Section 8.4) for information.

1.11 BEFLUX

Calculates the unconvolved energy flux for background datasets.

See the `EFLUX` command (Section 1.27) for information.

1.12 BGROUP

Applies grouping scheme to background data.

See the `GROUP` command (Section 1.42) for information.

1.13 BMCOUNTS

Calculates the sum of convolved model amplitudes for background datasets.

See the `MCOUNTS` command (Section 1.52) for information.

1.14 BSTATERRORS

Defines the statistical errors for background data.

See the `STATERRORS` command (Section 8.4) for information.

1.15 BSYSERRORS

Defines an expression or file to be used to specify the *systematic* errors for background data.

```
sherpa> BSYSERRORS [<dataset range> | ALLSETS] [ID] = <errorExpr>
```

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1. The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The error expression, <errorExpr>, may be composed of one or more (algebraically-combined) of the following elements:

Component:	Description:
BACK	An input background dataset
numericals	Numerical values
operators	+ - * / ( )

A few things to note:

- Background systematic error expressions cannot be defined without first reading in a background dataset.
- The input systematic error for a given bin is first multiplied by the datum in that bin; the resulting product is added in quadrature with the statistical error to yield the overall error for that bin.
- The background systematic errors can be specified in an input file; see `READ BSYSERRORS` (Section 1.64) for more information.
- If the background systematic errors are not defined using the `BSYSERRORS` command or input using `READ BSYSERRORS`, they are assumed to be zero.

For additional information, see the related command `BERRORS` (Section 1.8).

The background systematic errors are accessible to the *Sherpa*/S-Lang module user via the functions `get_bsyserrors` and `set_bsyserrors`.

Examples:

1. Define an expression to be used for the background systematic errors. In each bin, they will be computed as 0.1 times the background datum.

```
sherpa> DATA data.dat
sherpa> BACK back.dat
sherpa> BSYSERRORS = 0.1
```

1.16 BUNGROUP

Undo grouping applied to background data.

See the `UNGROUP` command (Section 1.86) for information.

1.17 BYE — EXIT — QUIT

Terminates the Sherpa program. Other equivalent termination commands are `EXIT` and `QUIT`.

```
sherpa> {BYE | EXIT | QUIT}
```

Examples:

1. Terminate the *Sherpa* program:

```
sherpa> BYE
Goodbye.
```

1.18 CAST

[REMOVED AS OF CIAO 3.0.2] Casts a dataset to single or double precision.

This command was removed from CIAO at version 3.0.2.

Sherpa internals have been updated to work with double-precision data only. This does not affect data entry: single-precision data are cast to double-precision, *etc.* The `CAST` command has been eliminated as it is now obsolete.

1.19 CLOSE

Closes the image display window.

```
sherpa> CLOSE IMAGE
```

where `IMAGE` is the imaging window (see the `IMAGE` command, Section 1.45.)

Note: Image display windows, as well as plotting display windows, may be closed simply by closing the window with a mouse click.

See the *Sherpa* Chapter for further information regarding data display capabilities within Sherpa.

Examples:

1. Open and close the imaging window:

```
sherpa> OPEN IMAGE
sherpa> CLOSE IMAGE
```

1.20 COORD

Specifies the coordinate system for use in fits to 2-D images.

```
sherpa> COORD [<dataset range> | ALLSETS] [<arg>]
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset number 1.

<**arg**> is either:

Argument	**Coordinate System**
{IMAGE | LOGICAL}	Logical coordinates: the bin numbers (1, 2, 3...)
PHYSICAL	Physical coordinates: a linear transformation of logical coordinates.
{WCS | WORLD}	World coordinates (RA,dec).

Table 1.4: COORD Command Arguments

Issuing the COORD command with no argument causes *Sherpa* to display the current setting for the specified dataset number(s).

Note that if, *e.g.*, a source model is defined *before* a switch of coordinate systems, it will be necessary either to issue the GUESS command (Section 1.43) to reset the initial parameter values and their ranges to more appropriate values, or to reset the values by hand.

Examples:

1. Fit data in physical coordinates:

```
sherpa> DATA example_img2.fits
sherpa> NOTICE FILTER "circle(425,343,30)"
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = GAUSS2D[g]
sherpa> COORD
Coordinate setting for dataset 1: logical
sherpa> COORD PHYSICAL
sherpa> GUESS SOURCE
sherpa> COORD
Coordinate setting for dataset 1: physical
sherpa> g.fwhm = 1
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 1289.82
 LVMQT: final statistic value = 1176.13 at iteration 12
             g.fwhm  2.39494
             g.xpos  4010.48
```

```
g.ypos  3927.48
g.ampl  40.3545
```

1.21 COVARIANCE

Computes covariance matrices, and provides an estimate of confidence intervals for selected thawed parameters.

```
sherpa> COVARIANCE [<dataset_range> | ALLSETS] [ <arg_1> , ... ]
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to estimate limits using data from all appropriate datasets.

The command-line arguments <arg_n> may be:

Argument	**Description**
`<sherpa_modelname>.{<paramname> | <#>}`	A specified model component parameter (*e.g.*, `GAUSS.pos`).
`<modelname>.{<paramname> | <#>}`	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.5: `COVARIANCE` Command Arguments

The user may configure `COVARIANCE` via the *Sherpa* state object structure `cov`. The current values of the fields of this structure may be displayed using the command `print(sherpa.cov)`, or using the more verbose *Sherpa*/S-Lang module function `list_cov()`.

The structure field is:

Field	**Description**
`sigma`	Specifies the number of σ (*i.e.*, the change in statistic).

Table 1.6: `cov` Structure Field

Field values may be set using directly, *e.g.*,

```
sherpa> sherpa.cov.sigma = 2.6
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_cov()`.

The confidence interval estimates are computed quickly, as described below, but are generally more accurate than those found using the command `UNCERTAINTY` (Section 1.85); see also `PROJECTION` (Section 1.61).

Because `COVARIANCE` estimates confidence intervals for each parameter independently, the relationship between `sigma` and the change in statistic value ΔS can be particularly simple: $\sigma = \sqrt{\Delta S}$ for statistics sampled from the χ^2 distribution and for the Cash statistic, and is approximately equal to $\sqrt{2\Delta S}$ for fits based on the general log-likelihood.

Confidence	σ	$\Delta\chi^2$	$\Delta \log \mathcal{L}$
68.3%	1.0	1.00	0.50
90.0%	1.6	2.71	1.36
95.5%	2.0	4.00	2.00
99.0%	2.6	6.63	3.32
99.7%	3.0	9.00	4.50

Table 1.7: Confidence Intervals for the `covariance` command

There are a number of computations and outputs associated with the `COVARIANCE` command:

First, an estimate of the information matrix is made. This is the matrix of second derivatives of the fit statistic at the best-fit point, or mode:

$$I_{i,j} \equiv \frac{1}{2}\frac{\partial^2\chi^2}{\partial p_i \partial p_j} \quad \text{or} \quad \frac{1}{2}\frac{\partial^2 C}{\partial p_i \partial p_j} \quad \text{or} \quad \frac{\partial^2 B}{\partial p_i \partial p_j}\,, \tag{1.1}$$

where C and B represent the `CASH` and `BAYES` statistics, and p_i represents the i^{th} thawed parameter.

The inverse of the information matrix provides an estimate of the covariance matrix:

$$C_{i,j} = I^{-1}_{i,j}\,. \tag{1.2}$$

The diagonal values of this matrix provide the variances; the $n\sigma$ confidence intervals are given by $n\sqrt{V_i}$. (For further details on how to interpret the covariance matrix, see, *e.g.*, pp. 198-199 of Eadie *et al.* 1971, *Statistical Methods in Experimental Physics* [Amsterdam: North-Holland].)

Output files include the information and covariance matrices, along with the eigenvectors and eigenvalues of the covariance matrix. These are recorded in three temporary ASCII files in the `$ASCDS_WORK_PATH` directory: **`ascfit.inf_matrix.`<`number`>**, **`ascfit.cov_matrix.`<`number`>**, and **`ascfit.eig_vector.`<`number`>**, where <**`number`**> refers to the process ID (pid) number for the *Sherpa* run. These files may be saved by copying them from the `$ASCDS_WORK_PATH` directory during the *Sherpa* session. The files are deleted from the working directory when the *Sherpa* session is finished.

1.21.1 Caveats

An estimated confidence interval is accurate if and only if:

1. the χ^2 or $\log\mathcal{L}$ surface in parameter space is approximately shaped like a multi-dimensional paraboloid, and

2. the best-fit point is sufficiently far from parameter space boundaries.

One may determine if these conditions hold by plotting the fit statistic as a function of each parameter's values (the curve should approximate a parabola) and by examining contour plots of the fit statistics made by varying the values of two parameters at a time (the contours should be elliptical, and parameter space boundaries should be no closer than approximately 3σ from the best-fit point).

Note that these conditions are the same as those which dictate whether the use of PROJECTION (Section1.61) will yield accurate errors. While PROJECTION is more general (*e.g.* allowing the user to examine the parameter space away from the best-fit point), it is in the strictest sense no more accurate than COVARIANCE for determining confidence intervals.

If either of the conditions given above does not hold, **then the output from COVARIANCE may be meaningless except to give an idea of the scale of the confidence intervals**. To accurately determine the confidence intervals, one would have to reparameterize the model, or use Monte Carlo simulations or Bayesian methods.

Examples:

1. List the current and default values of the cov structure, and restore the default values:

```
sherpa> sherpa.cov.sigma = 5
sherpa> list_cov()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
sigma             5               1                    Number of sigma
sherpa> restore_cov()
sherpa> list_cov()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
sigma             1               1                    Number of sigma
```

2. Determine the covariance matrix and errors for all thawed parameters:

```
sherpa> DATA example1a.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = GAUSS1D[12]
sherpa> FIT
...
sherpa> set_verbose(2)
sherpa> COVARIANCE
Information Matrix (Second Derivatives of Fit Statistic):

      p.c0            p.c1            p.c2            p.c3
        0.447924        1.16116         4.64118         24.2449
         1.16116        4.64109         24.2449         146.113
         4.64118        24.2449         146.113           954.8
         24.2449        146.113           954.8         6560.89
```

```
Eigenvectors (Principal Axes) of the Covariance Matrix:

      p.c0           p.c1           p.c2           p.c3
        0.646594       0.747372       0.152771      0.00368201
       -0.734474        0.55588       0.388661       0.0221011
        0.205453      -0.361973       0.897769        0.144137
      -0.0159319      0.0375378      -0.140052        0.989304

Eigenvalues of the Covariance Matrix:

         159.066        3.55303       0.118346     0.000149179

Covariance Matrix (Inverse of Information Matrix):

      p.c0           p.c1           p.c2           p.c3
         68.4903       -74.0584        20.1862        -1.54147
        -74.0584        86.9244       -24.6767         1.92902
         20.1862       -24.6767        7.27528       -0.583802
        -1.54147        1.92902      -0.583802       0.0478489

Covariance Matrix Determinant (Product of Eigenvalues): 0.00997785


Computed for covariance.sigma = 1
        --------------------------------------------------------
        Parameter Name      Best-Fit Lower Bound     Upper Bound
        --------------------------------------------------------
             p.c0         -0.303712  -8.27589       +8.27589
             p.c1          0.611953  -9.32332       +9.32332
             p.c2          0.790141  -2.69727       +2.69727
             p.c3         0.0184866  -0.218744      +0.218744
```

1.21.2 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `COVARIANCE` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `COVARIANCE` will not be valid unless the parameters are at their best-fit values.

1.22 CPLOT

Causes the specified 2-D data to be displayed, with a contour plot, via *ChIPS*.

```
sherpa> CPLOT [<num_plots>] <arg_1> [# [ID]] [<arg_2> [# [ID]] ...]
```

<`num_plots`> specifies the number of plotting windows to open within the *ChIPS* pane (default 1); that number sets the number of subsequent arguments. For each subsequent argument, `#` specifies the number of the dataset (default dataset number is 1), and the `ID` modifier is used for displaying background datasets, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The argument <`arg_n`> may be any of the following:

<table>
<tr><th>Argument</th><th>Displays</th></tr>
<tr><td>{{DATA | DCOUNTS} | {BACK | BDCOUNTS}}</td><td>The source|background data values</td></tr>
<tr><td>{ERRORS | BERRORS}</td><td>The estimated total errors for the source|background data values</td></tr>
<tr><td>{SYSERRORS | BSYSERRORS}</td><td>The assigned systematic errors for the source|background data values</td></tr>
<tr><td>{STATERRORS | BSTATERRORS}</td><td>The estimated statistical errors for the source|background data values</td></tr>
<tr><td>{{MODEL | MCOUNTS} | {BMODEL | BMCOUNTS}}</td><td>The (convolved) source|background model amplitudes</td></tr>
<tr><td>{DELCHI | BDELCHI}</td><td>The sigma residuals of the source|background fit: (data - model)/error</td></tr>
<tr><td>{RESIDUALS | BRESIDUALS}</td><td>The absolute residuals of the source|background fit: (data - model)</td></tr>
<tr><td>{RATIO | BRATIO}</td><td>The ratio (data/model) for source|background</td></tr>
<tr><td>{CHI SQU | BCHISQU}</td><td>The contributions to the χ^2 statistic from each source|background data point</td></tr>
<tr><td>{STATISTIC | BSTATISTIC}</td><td>The contributions to the current statistic from each source|background data point</td></tr>
<tr><td>{WEIGHT | BWEIGHT}</td><td>The statistic weight value assigned to each source|background data point</td></tr>
<tr><td>{FILTER | BFILTER}</td><td>The mask value (0|1) for each source|background data point</td></tr>
<tr><td><sherpa_model_stack></td><td>The (unconvolved) model amplitudes for the specified model stack (SOURCE, {BACKGROUND|BG}, or [B]NOISE)</td></tr>
<tr><td><model_stack></td><td>The (unconvolved) model amplitudes for the specified user-defined model stack</td></tr>
<tr><td><sherpa_modelname></td><td>The (unconvolved) amplitudes of the specified model component (e.g., GAUSS2D)</td></tr>
<tr><td><modelname></td><td>The (unconvolved) amplitudes of the specified model component (e.g., g)</td></tr>
<tr><td>{EXPMAP | BEXPMAP}</td><td>The unfiltered source|background 2-D EXPMAP file contents</td></tr>
<tr><td>{PSF | BPSF}</td><td>The unfiltered source|background 2-D PSF file contents</td></tr>
</table>

Table 1.8: `CPLOT` Command Arguments

If there is no open plotting window when an CPLOT command is given, one will be created automatically.

If one issues the CPLOT following filtering, note the following: arbitrarily filtered data cannot be passed from *Sherpa* to *ChIPS* for display; the data grid must be rectangular. Therefore, contouring proceeds in three steps: (1) the smallest possible rectangle is drawn around the noticed data; (2) within this rectangle, the <data to image> is tranformed to <data to image> * filter; and (3) these transformed data are sent off to *ChIPS* for display.

The appearance of plots generated with this command can be changed by modifying the fields of certain state objects. See the ahelp for *Sherpa* or for sherpa.plot for more information.

NOTE: all *ChIPS* commands may be used from within *Sherpa* to modify plot characteristics. In order to view these changes, the REDRAW command must be issued.

See the display chapter (Chapter 2) for more information regarding data display capabilities, including modifying various plot characteristics.

Examples:

1. Display 2-D data with a contour plot:

```
sherpa> DATA 3 example2Da.dat ASCII 1 2 3
sherpa> CPLOT DATA 3
```

   The CPLOT command plots dataset number 3 as a contour plot. Dataset number 3 must be a 2-D dataset.

2. Display 2-D datasets with contour plots in multiple windows:

```
sherpa> DATA 1 example2Db.dat ASCII 1 2 3
sherpa> CPLOT 2 DATA 1 DATA 3
```

   This command displays a contour plot of dataset number 1 (example2Db.dat) in the first window, and a contour plot of dataset number 3 (example2Da.dat) in the second window.

1.23 CREATE

Establishes a model component and its parameters, for use in the current *Sherpa* session.

```
sherpa> [CREATE] <sherpa_modelname>
```

(The brackets around CREATE indicate that its use is optional; see examples below.) In addition, *Sherpa* provides the user with the ability to both establish a model component, and to assign to it an arbitrary name:

```
sherpa> [CREATE] <sherpa_modelname>[<modelname>]
```

where <sherpa_modelname>is the *Sherpa* default model name (or an *XSPEC* model name, with the prefix 'xs' attached), and <`modelname`> is the name being given to the model component by the user. Note that <`modelname`> *must be enclosed in brackets*, [ ].

Sherpa's ability to assign arbitrary names to model components allows the user to establish multiple independent models of the same type during a single session, and is a valuable feature of the software. Note that an assigned model name can be any arbitrary string, except a string that is already a *Sherpa* command.

By default, *Sherpa* will prompt the user for the initial model component parameter values. (Parameter prompting can be turned off using the command `PARAPROMPT OFF`.) At the model parameter prompt, the user may either:

- Hit the <`RETURN`> key to accept the given initial parameter value. Note that if a dataset has not been input, then the initial parameter values are arbitrary default values. However, if a dataset has been input, then the initial parameter values are estimates based on the filtered data and the defined instrument model (see `AUTOEST`for more information).
- Hit the <`RETURN`> key to accept the given initial parameter value, and subsequently set the parameter value and range using one of the syntax options whose descriptions may be found in `PARAMSET` (see `CREATEPARAMSET`).
- Hit the <`RETURN`> to accept the given initial parameter value, and subsequently set the parameter value using the `GETX` or `GETY` command, which allows the user to assign the value interactively from a plot.
- Enter ,`-1` to accept the given initial parameter value, and to set the parameter to be frozen. Note that there *cannot* be a space between the `,` and the `-1`.
- Enter <`value`>`:`<`min`>`:`<`max`>[`:`<`delta`>][`,-1`], where:

Argument	Description
<`value`>	Model parameter initial value.
<`min`>	Minimum value for that parameter range.
<`max`>	Maximum value for that parameter range.
<`delta`>	Specifies initial parameter step size.
`,-1`	Sets the parameter to be frozen.

Note that the default setting for `delta` is 1% of the parameter value. Otherwise, specifying `delta` will set the initial parameter step size to an absolute value. The parameter step size is used by the optimization method to determine where to sample parameter space. As such, if the value of a parameter is known well, then specifying a small `delta` may lead to a substantially faster and better fit.

Note that this colon separated list need not include all of the elements, but it does need to maintain the proper sequential order, and also include the colons around skipped elements. Also, there cannot be a space between the `,` and the `-1`.

The model components that have been established in the current *Sherpa* session, and their parameter information, may be listed with the command `SHOW MODELS`. Note however that information about model parameter delta settings is currently not returned by `SHOW`. Information about model parameter delta settings is included when using `SAVE`.

For more information on:	Type:
Setting parameter values and ranges	`ahelp paramset`, `ahelp createparamset`
Linking parameter values	`ahelp linkparam`
Freezing and thawing model parameters	`ahelp freeze`, `ahelp thaw`
How parameter value estimates are made	`ahelp autoest`, `ahelp guess`
Controlling model integration	`ahelp integrate`
Creating model expressions	`ahelp modelexpr`
Creating model stacks	`ahelp modelstack`
Creating nested models	`ahelp nestedmodels`
Creating joint-mode models	`ahelp jointmode`

Examples:

1. Establish a model component

```
sherpa> CREATE POLY
POLY.c0 parameter value [1]
POLY.c1 parameter value [0]
POLY.c2 parameter value [0]
POLY.c3 parameter value [0]
POLY.c4 parameter value [0]
POLY.c5 parameter value [0]
POLY.c6 parameter value [0]
POLY.c7 parameter value [0]
POLY.c8 parameter value [0]
POLY.offset parameter value [0]
```

The command `CREATE POLY` establishes the *Sherpa* model`POLY` as a model component available for use in the current *Sherpa* session. Note that after issuing this command, the user is prompted for the initial model parameter values. In this example, the user accepted the given initial values for all of the parameters, using the <`RETURN`> key. The following command is equivalent:

```
sherpa> POLY
```

2. Establish a model component, and assign it a name:

```
sherpa> CREATE POLY[modela]
modela.c0 parameter value [1]
modela.c1 parameter value [0]
modela.c2 parameter value [0]
modela.c3 parameter value [0]
modela.c4 parameter value [0]
modela.c5 parameter value [0]
modela.c6 parameter value [0]
modela.c7 parameter value [0]
modela.c8 parameter value [0]
modela.offset parameter value [0]
```

The command `CREATE POLY`[`modela`] establishes the *Sherpa* model component `POLY`, and assigns to it the name `modela`. Note that after issuing this command, the user is prompted for the initial model parameter values. In this example, the user accepted the given initial values for all of the parameters, using the <`RETURN`> key. The following command is equivalent:

```
sherpa> POLY[modela]
```

3. Establish a model component, assign it a name, set parameter values and ranges including initial parameter step size, and freeze a parameter:

```
sherpa> GAUSS[modelb]
modelb.fwhm parameter value [10] 2.0:1:10:0.5
modelb.pos parameter value [0] 1:0.1:
modelb.ampl parameter value [1] 3::100:2,-1
```

This example establishes and assigns the name modelb to the *Sherpa* model component `GAUSS1D`. Note that in this example the user entered an initial value of 2.0, a minimum range value of 1, a maximum range value of 10, and an initial parameter step size `delta` of 0.5 for parameter `fwhm`; entered a value of 1, and a minimum range value of 0.1 for parameter `pos`; and entered a value of 3, a maximum range value of 100, and an initial parameter step size of 2 for parameter `ampl`, and froze this parameter.

4. Establish a model component, assign it a name, set parameter values and ranges, and freeze a parameter:

```
sherpa> GAUSS[modelbb]
modelbb.fwhm parameter value [10] 2.0:1:10
modelbb.pos parameter value [0] 1:0.1:
modelbb.ampl parameter value [1] 3::100,-1
```

This example establishes and assigns the name `modelb` to the *Sherpa* model component `GAUSS1D`. Note that in this example the user entered an initial value of 2.0, a minimum range value of 1, and a maximum range value of 10 for parameter `fwhm`; entered a value of 1, and a minimum range value of 0.1 for parameter `pos`; and entered a value of 3, and a maximum range value of 100 for parameter `ampl`, and froze this parameter (note that after the maximum range value, specification of initial parameter step size `delta` is optional and may be omitted).

5. Establish an *XSPEC* model component, assign it a name, and freeze a parameter:

```
sherpa> XSBBODY[modeld]
modeld.kT parameter value [3]
modeld.norm parameter value [1] ,-1
```

This command establishes and assigns the name `modeld` to the *XSPEC* model component `bbody`. In this example, the user accepted the given initial values for all of the parameters, using the <`RETURN`>, and set parameter `norm` to be frozen.

6. Turn off model parameter prompting; establish a model component and assign it a name:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
```

The first command turns off prompting for model parameter values. The second command establishes the *Sherpa* model component `POW`, and assigns to it the name `modelc`. Since parameter prompting was turned off, the model is automatically established using the given initial values for all of the parameters.

7. Establish two independent model components of the same type:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> POW[modelc]
modelc.gamma parameter value [0] 1.0
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
sherpa> POW[modelc2]
modelc2.gamma parameter value [0] 2.0
modelc2.ref parameter value [1]
modelc2.ampl parameter value [1]
```

The third command establishes the *Sherpa* model component `POWLAW1d`, and assigns to it the name `modelc`. In this example, the user set parameter `gamma` of model component `modelc` to the value of 1.0. The last command establishes another *Sherpa* model component `POWLAW1d`, and assigns to it the name `modelc2`. In this example, the user set parameter `gamma` of model component `modelc2` to the value of 2.0. Note that `model` components `modelc` and `modelc2` are independent of one another.

1.24 DATA

Inputs the contents of one or more source data files.

```
sherpa> DATA [#] <filespec> [, [#] <filespec>,...]
```

where `#` specifies the number of the dataset to be associated with this source data file (default dataset number is 1).

Alternative means of reading in source datasets involve using the `load` functions of the *Sherpa*/S-Lang module (*e.g.*, `load_pha`).

The help file for the `READ` (Section 1.64) command explains the <filespec> definition and has a listing of allowed file types; see also the related commands `SOURCE` (Section 1.77), `ERRORS` (Section 1.30), and `SETDATA` (Section 1.75).

1.24.1 Using Data Model Filters

This command is able to take any Data Model virtual file specification (see "ahelp dmsyntax"). If you can do

```
unix% dmcopy "infile.fits[spec 1][spec 2]" outfile.fits
```

you can also do

```
sherpa> data "infile.fits[spec 1][spec 2]"
```

This is especially useful when working with very large files. For example:

```
sherpa> data "evt.fits[bin sky=4][opt mem=100]"
```

bins the event file by a factor of four and allocates additional memory. A similar command (omitting the binning factor) can be used to read in an image.

Examples:

1. Input an ASCII data file having a `.dat` extension name:

   ```
   sherpa> DATA example.dat
   ```

   This command reads the first two columns of the ASCII data file `example.dat`, as dataset number 1.

2. Input an ASCII data file not having a .dat extension name:

   ```
   sherpa> DATA example.qdp ASCII 1 2
   ```

   This command reads columns 1 and 2 of the ASCII data file `example.qdp`, as dataset number 1.

3. Input a FITS image data file:

   ```
   sherpa> DATA 3 data/example_img.fits FITS
   ```

   This command reads the FITS image `data/example_img.fits`,as dataset number 3.

1.25 DATASPACE

Creates a data grid on which models may be evaluated.

```
sherpa> DATASPACE [#] (<range> [, <range>, ...]) [HISTOGRAM]
```

where `#` specifies the number of the dataset to be associated with the dataspace (default dataset number is 1); <range> is defined below; and `HISTOGRAM` tells *Sherpa* to define bins (with lower and upper boundaries) rather than single gridpoints.

`<range>` = `<start>:<stop>:<delta>`, where

Argument	Definition
<`start`>	The start (minimum) value for the grid.
<`stop`>	The stop (maximum) value for the grid.
<`delta`>	The step size between gridpoints.

Table 1.11: Input Arguments for `DATASPACE`

If `HISTOGRAM` is specified, the models will be evaluated by integrating over bins of width <`delta`>; otherwise, models will be evaluated at points on the specified grid.

Note: `HISTOGRAM` must be specified in order to evaluate *XSPEC* models additive models (*e.g.*, `xsbremss`).

A dataspace may also be defined using the *Sherpa*/S-Lang module functions `set_axes` and `set_baxes`. (In CIAO 3.0, `set_baxes` is the only means by which *background* dataspaces may be defined.)

Examples:

1. Set a 1-D value range on which a source model may be evaluated:

   ```
   sherpa> DATASPACE (1:5:1)
   ```

   This command sets the value range, from values 1 through 5, with a step-size of 1, over which a source model may be evaluated.

2. Set a 2-D value range on which a source model may be evaluated:

   ```
   sherpa> DATASPACE (1:5:1,1:2:1)
   ```

   This command sets the value ranges, for two dimensions, over which a source model may be evaluated.

3. Set a 1-D value range on which a source model may be evaluated, for dataset number 2:

   ```
   sherpa> DATASPACE 2 (1:10:1)
   ```

   This command sets the value range, from values 1 through 10, with a step-size of 1, over which a source model may be evaluated, for dataset number 2.

1.26 ECHO

Turns on/off reporting of user keystrokes.

```
sherpa> ECHO {ON | OFF}
```

The ECHO command can be used to create a log of an entire session - including screen output - when run from a script.

By default, reporting of user keystrokes is turned off.

Examples:

1. Turn on reporting of user keystrokes; turn off reporting of user keystrokes:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> ECHO ON
Echo is on
ECHO ON
sherpa> POLY[modela]
POLY [ modela ]
sherpa> ECHO OFF
Echo is off
sherpa> GAUSS[modelb]
sherpa>
```

1.27 EFLUX

Calculates the unconvolved energy flux for source or background datasets.

```
sherpa> [B]EFLUX [# [ID]] [{(<value>) | (<min>:<max>) | (<region descriptor>)} ] \
[ {(<model component>) | (<model stack>)} ]
```

EFLUX is used for computing source energy fluxes, while BEFLUX is used for computing background energy fluxes.

specifies the dataset over which the source model is evaluated. The ID modifier is used only for computing background fluxes, and then if and only if the *Sherpa* state object variable multiback is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The ID modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The default is to compute energy fluxes for all appropriate datasets (*i.e.*, those for which source/background expressions have been defined). The flux may be computed at one energy/wavelength, over a range of energies/wavelengths, or within a 2-D region, with the default being to compute the flux the total available range.

The flux may also be computed for individual model components, or for previously defined model stacks, with the default being to compute the flux using all model components in the SOURCE or BACKGROUND expression.

A source or background model stack must be defined before a respective flux can be computed; see the SOURCE, Section 1.77, and BACKGROUND, Section 1.9, commands. This is true even if one computes the flux of an individual model component or of models defined in a model stack. (This limitation will be removed in a future version of *Sherpa*.)

For 1-D data, if:

- one energy (or wavelength) <value> is given, then the energy flux at that energy (or wavelength) is returned in units of ergs/cm^2/sec/keV (or ergs/cm^2/sec/Å).
- an energy or wavelength range is given with the <`min`> and <`max`> arguments, then the total integrated energy flux over the given range is returned in units of ergs/cm^2/sec.
- no value or range is specified, then the total integrated energy flux over the full range of the dataset is returned in units of ergs/cm^2/sec.

For 1-D data, if an instrument model is not used, the units are (perhaps incorrectly) assumed to be counts, or counts per bin-width. See the *Note on Units* below.

For 2-D data, if a region descriptor is given, then the total integrated energy flux within that region is returned; otherwise, the integration is carried out over the entire input image.

Note on Units: In its current incarnation, *Sherpa* has no explicit knowledge of data or model units. The units displayed with computed fluxes are defaults, generally correct for standard analyses of 1-D PHA energy/wavelength spectra (*XSPEC*-like analyses). They may be incorrect for non-standard analyses, or for analyses of 2-D spatial images with exposure maps, *etc.* The correct units can be determined by working backwards from the data, taking into account the exposure time, the units of the instrument model, the bin units, *etc.*

Tip: To perform background subtraction in *Sherpa*, the command `SUBTRACT`, Section 1.80, must be issued; this is in contrast to *XSPEC*, which performs background subtraction automatically.

The energy flux may be computed using the *Sherpa*/S-Lang module functions `get_eflux` and `get_beflux`.

Examples:

1. Calculate the integrated energy flux over the full energy range:

```
sherpa> EFLUX
Flux for source dataset 1: 1.94551e-13 ergs/cm**2/s
```

2. Calculate the integrated background photon flux over the range 2 to 10 keV:

```
sherpa> BEFLUX (2.0:10.0)
Flux for background dataset 1: 3.59906e-13 ergs/cm**2/s
```

3. Calculate the energy flux at a single energy (2.0 keV) for the power-law component of a source expression:

```
sherpa> SOURCE 1 = XSWABS[A] * POW[P]
sherpa> EFLUX 1 (2.0) P
Flux for source dataset 1: 1.73452e-14 ergs/cm**2/s/keV
```

4. Calculate the total energy flux over the energy range 2.0 to 4.0 keV for a model stack:

```
sherpa> FOO = POW[P] + GAUSS[G]
sherpa> SOURCE 1 = XSWABS[A] * FOO
sherpa> EFLUX 1 (2:4) FOO
Flux for source dataset 1: 2.46491e-13 ergs/cm**2/s
```

1.28 EQWIDTH

Computes the equivalent width of an emission or absorption line in source or background data.

```
sherpa> [B]EQWIDTH [# [ID]] (<continuum_stack>,
<continuum_plus_line_stack>)
```

`EQWIDTH` is used for computing equivalent widths in source datasets, while `BEQWIDTH` is used for computing equivalent widths in background datasets.

`#` specifies the dataset over which the source model is evaluated. The `ID` modifier is used only for computing background dataset equivalent widths, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

<`continuum_stack`> represents one or more models (or user-defined model stacks) that describe the continuum, while <`continuum_plus_line_stack`> represents two or more models (or user-defined model stacks) that describe the continuum plus line.

To compute the equivalent width, an integral over the energy/wavelength range of the dataset is performed. At each point, the <`continuum_stack`> (C) and <`continuum_plus_line_stack`> ($[C + L]$) are evaluated; the integrand is then ($[C + L] - C$).

The models are specified on-the-fly in the same manner that `SOURCE` (Section 1.77) or `BACKGROUND` (Section 1.9) model stacks are defined, as algebraic combinations of previously defined model components. The examples below illustrate this point. The user must specify the models in this manner, because (a) *Sherpa* cannot identify whether a particular model component should be associated with the continuum or with the line, and (b) multiple line components may be specified in the `SOURCE` model stack, so that it cannot be used in place of <`continuum_plus_line_stack`> to compute equivalent widths.

Equivalent widths may also be computed using the *Sherpa*/S-Lang module functions `get_eqwidth` and `get_beqwidth`.

Examples:

1. Model a continuum and emission line complex using a power-law and normalized Gaussian, then compute the equivalent width:

```
sherpa> SOURCE = POW[cont]+NGAUSS[eline]
```

```
...
sherpa> FIT
...
sherpa> EQWIDTH 1 (cont,cont+eline)
EW = 0.535073 keV
```

2. Define continuum and line model stacks; use these in the equivalent width calculation:

```
sherpa> BBODY[modela]
sherpa> POWLAW1D[modelb]
sherpa> NGAUSS[line1]
sherpa> NGAUSS[line2]
sherpa> NGAUSS[line3]
...
sherpa> CONT = modela+modelb
sherpa> ELINE = line1
sherpa> SOURCE = CONT + ELINE + line2 + ...
sherpa> FIT
...
sherpa> EQWIDTH 1 (CONT,CONT+ELINE)
EW = 0.454946 keV
```

1.29 ERASE

Erases user inputs and settings, and/or model components.

```
sherpa> ERASE [{ALL | <sherpa_modelname> | <modelname> |
<model_stack> | DATA [#] | BACK [# [ID]]}]
```

where `#` specifies the number of the dataset to be erased (default dataset number is 1). The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

`ERASE ALL` causes the following to be erased from the current *Sherpa* session:

- All source and background datasets.
- All defined model components and model stacks.

Note that `ERASE ALL` does not cause the optimization method, statistic choice, parameter prompting, or variables within the state object to be reset to default values. `RESET` is a related command that may be used to restore program settings and parameter values.

`ERASE` <sherpa_modelname> and `ERASE` <modelname> cause the specified single model component to be erased. Note, however, that a single component cannot be erased if it is part of a model stack definition (*e.g.*, `SOURCE`). Clear the stack first (by issuing, *e.g.*, the command `SOURCE =`), then issue the `ERASE` command.

`ERASE <model_stack>` causes the specified single user-defined model stack to be removed; like above, it cannot be erased if it is part of another stack's definition.

Examples:

1. Remove a model component:

```
sherpa> DATA example.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
sherpa> SHOW
Current Data Files:
Data 1: data/example.dat ascii 1 2.
Total Size: 4 bins (or pixels)
Dimensions: 1
Total counts (or values): 31

Optimization Method: Powell

Statistic: Chi-Squared Gehrels

Current Models are:

Current Composite Models are:

Current Model Components are:
powlaw1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          1        -10         10
 2    ref frozen          1          1          4
 3   ampl thawed          3     0.0300        300
sherpa> ERASE modelc
```

In this example, a dataset is input and a model component called `modelc` is established. The `ERASE` command then removes this model component.

2. Erase a background dataset:

```
sherpa> DATA example2.pha
...
Background data are being input from:
  <directory path>/example2_bkg.pha
sherpa> SHOW
...
-----------------
Input data files:
-----------------

Data 1: example2.pha pha.
```

```
...

  Background 1: <directory path>/example2_bkg.pha pha.
...
sherpa> ERASE BACK
sherpa> SHOW
...
-----------------
Input data files:
-----------------

Data 1: example2.pha pha.
...
sherpa>
```

In this example, the background dataset was automatically loaded into *Sherpa* because the source dataset PHA file contained the header keyword BACKFILE.

3. Remove all user inputs and user-defined settings:

```
sherpa> METHOD GRID
sherpa> STATISTIC BAYES
sherpa> ERASE ALL
sherpa> SHOW
Current Data Files:

Optimization Method: Grid

Statistic: Bayes

Current Models are:

Current Composite Models are:

Current Model Components are:
```

This example illustrates that ERASE ALL does not return either the optimization method or the statistic to the default setting (the default method is LEVENBERG-MARQUARDT; the default statistic is CHI GEHRELS).

1.30 ERRORS

Defines an expression to be used to specify the *statistical* errors for source data.

```
sherpa> ERRORS [<dataset range> | ALLSETS] = <errorExpr>
```

where <dataset range> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The error expression, <`errorExpr`>, may be composed of one, or more (algebraically-combined) of the following elements:

Component:	**Description:**
`DATA`	An input dataset
`numericals`	Numerical values
`operators`	`+ - * / ( )`

A few things to note:

- Error expressions cannot be defined without first reading in a source dataset.
- The errors can be specified in an input file; see `READ ERRORS` (Section 1.64) for more information.
- If the errors are not defined using `ERRORS` or input using `READ ERRORS`, then they are computed as defined for the current statistic; see the documentation on the command `STATISTIC` (Section 1.79).

See the related command `SYSERRORS` (Section 1.82).

The errors are accessible to the *Sherpa*/S-Lang module user via the functions `get_errors` and `set_errors`.

Examples:

1. Define an expression to be used for the errors. They are set to one-half of the data value in each bin.

```
sherpa> DATA example.dat
sherpa> ERRORS = (0.5)*(DATA)
```

2. Define an expression to be used for the errors. They are set to be the sum of the datum and 2.7 in each bin.

```
sherpa> DATA example.dat
sherpa> ERRORS 1 = DATA + 2.7
```

1.31 EXIT — BYE — QUIT

Terminates the *Sherpa* program.

See the `BYE` command (Section 1.17).

1.32 FAKEIT

Creates a simulated 1-D dataset.

```
sherpa> FAKEIT [#]
```

`#` specifies the number of the dataset to be simulated (default dataset number is 1).

The `FAKEIT` command creates a simulated 1- or 2-D dataset. It cannot be successfully issued without first:

- defining a `SOURCE` model stack; and
- defining a grid over which to evaluate the model stack.

The user may define the grid in one of three ways:

- reading in a dataset (in which case, the faked data will replace the read-in amplitudes);
- specifying a `DATASPACE`; or
- defining an instrument stack that contains a response matrix file (RMF), from which a dataspace may be inferred; note that an instrument stack be defined automatically if PHA data are input, and the PHA data file header keywords `RESPFILE` and/or `ANCRFILE` point to existing files.

If one defines a dataspace and plans to use *XSPEC* models, then one must use the `HISTOGRAM` modifier to the `DATASPACE` command, since *XSPEC* models expect the dataset to be binned. Otherwise, the simulated data set may be either binned or unbinned.

Other, optional information may be input before `FAKEIT` is run.

1. A background dataset (see the command `BACK`) *or* a background model stack (`BACKGROUND`). If PHA data are input, a background dataset will be automatically input if the header keyword `BACKFILE` points to an existing file.
2. Values for the `FAKEIT` parameters `TIME` and `BACKSCALE`. If PHA data are input, `TIME` and `BACKSCALE` will have default settings corresponding to the values of the header keywords `EXPTIME` and `BACKSCAL`. These may of course be changed; see below.

Values of the `FAKEIT` parameters may be set as follows:

```
sherpa> FAKEIT TIME = <time>
sherpa> FAKEIT BACKSCALE = <backscale>
```

where <`time`> is in seconds, <`backscale`> is a dimensionless number.

Background data and/or models are treated as follows in `FAKEIT`:

1. *If a background model stack is defined*, it is evaluated on the source data grid, and the resulting background amplitudes are added to the source amplitudes (taking into account differences in exposure time and backscale). Faked data are then sampled given the sum. If background data exist, they are not altered. If the source dataset was background-subtracted prior to the command `FAKEIT` being issued, it will *not* be background-subtracted afterwards.

2. *If no background model stack is defined, and the data are background-subtacted*, then the source model stack is evaluated directly, and the new, faked data are background-subtracted. Note that subsequently issuing an `UNSUBTRACT` command in CIAO 3.0 is unwise, because as an artifact of the algorithm the unsubtracted data will not be integer counts data.
3. *If no background model stack is defined, and the data are not background-subtacted*, then the source model stack is evaluated directly, and the (properly scaled) background data are added to the faked data.

`FAKEIT` creates a new dataset and keeps it in memory (the *Sherpa* number assigned to this dataset is specified with `#` in the `FAKEIT` command). This new dataset may then be treated as though it were a dataset that the user had read from a file (*e.g.*, it may be plotted using the `LPLOT DATA [#]` command; it may be used in a fit, *etc.*). Note that if the user has previously read a dataset, it will be overwritten with the new dataset created by `FAKEIT`.

Note that in CIAO 3.0, there is no facility for faking background spectra (*i.e.*, there is no `BFAKEIT` command).

Examples:

1. Simulate a dataset, with an instrument model:

```
sherpa> RSP[instrumentA](data/example2.rmf,data/example2.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT = instrumentA
```

The above commands define an instrument model named `instrumentA`, using the RMF and ARF files `data/example2.rmf` and `data/example2.arf` respectively. Next, a background data file is input:

```
sherpa> BACK data/example2_bkg.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
```

The `FAKEIT` parameters are then set:

```
sherpa> FAKEIT TIME = 33483.2
sherpa> FAKEIT BACKSCALE = 0.0441895
sherpa> SHOW FAKEIT
Fakeit exposure time: 33483.2 seconds.
Fakeit backscale: 0.0441895
```

And, a simple source model expression is defined:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POW[modela]
sherpa> modela.gamma=2
sherpa> modela.ampl.min=0.000001
sherpa> modela.ampl=0.0003
sherpa> modela.ref=1
```

Finally, a simulated dataset is created:

```
sherpa> FAKEIT
FAKEIT: The current background data have been added to the faked spectrum.
```

This simulated dataset may be plotted, and written as PHA and ASCII files:

```
sherpa> LPLOT DATA
sherpa> WRITE DATA sim1.pha PHA
Write X-Axis: Energy (keV)  Y-Axis: Counts
sherpa> WRITE DATA sim1.dat ASCII
Write X-Axis: Energy (keV)  Y-Axis: Flux (Counts/sec/keV)
```

2. Simulate a dataset, utilizing a previously input PHA file:

```
sherpa> ERASE ALL
sherpa> DATA data/example2.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example.rmf
ARF is being input from:
  <directory_path>/example.arf
Background data are being input from:
  <directory_path>/example_bkg.pha
sherpa> SHOW

Optimization Method: Levenberg-Marquardt
Statistic:           Chi-Squared Gehrels

-----------------
Input data files:
-----------------

Data 1: example2.pha pha.
Total Size: 95 bins (or pixels)
Dimensions: 1
Total counts (or values): 1688
Exposure: 33483.25 sec
Count rate: 0.050 cts/sec
Backscal: 0.044189

  Background 1: /data/simteste/Testing/sherpaTest/data/example2_bkg.pha pha.
  Total Size: 512 bins (or pixels)
  Dimensions: 1
  Total counts (or values): 2220
  Exposure: 108675.66 sec
  Count rate: 0.020 cts/sec
  Backscal: 0.044189
```

```
  The data are NOT background subtracted.


------------------------------
Defined analysis model stacks:
------------------------------

instrument source 1 = AutoReadResponse
instrument back 1 = AutoReadResponse

------------------------------------
Defined instrument model components:
------------------------------------

rsp1d[AutoReadResponse]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1    rmf string: "/data/simteste/Testing/sherpaTest/data/example2.rmf"
 2    arf string: "/data/simteste/Testing/sherpaTest/data/example2.arf"

sherpa> SHOW FAKEIT
Fakeit exposure time: 33483.2 seconds.
Fakeit backscale: 0.0441895
```

After inputting a PHA dataset, the `SHOW` command confirms that an instrument model has been automatically defined using RMF and ARF files. Also, a background data file has been automatically read. The `SHOW FAKEIT` command confirms that the input data file contained the exposure time, backscale, and areascale keywords pertinent to the observation. Next, a source model expression is defined, and then a simulated dataset is created and written:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POW[modelA]
sherpa> modelA.gamma=2
sherpa> modelA.ampl.min=0.000001
sherpa> modelA.ampl=0.0003
sherpa> modelA.ref=1
sherpa> FAKEIT
FAKEIT: The current background data have been added to the faked spectrum.
sherpa> WRITE DATA simA.pha PHA
Write X-Axis: Energy (keV)  Y-Axis: Counts
```

3. Simulate a second dataset:

```
sherpa> RSP[instrumentB](data/example2.rmf,data/example2.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT 2 = instrumentB
sherpa> BACK 2 data/example2_bkg.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
```

```
sherpa> FAKEIT 2 TIME = 66966.4
sherpa> FAKEIT 2 BACKSCALE = 0.0441895
sherpa> SOURCE 2 = POW[modelB]
sherpa> modelB.gamma=2
sherpa> modelB.ampl.min=0.000001
sherpa> modelB.ampl=0.0003
sherpa> modelB.ref=1
sherpa> FAKEIT 2
FAKEIT: The current background data have been added to the faked spectrum.
sherpa> LPLOT 2 DATA 1 DATA 2
sherpa> WRITE DATA 2 fakeit3.pha PHA
Write X-Axis: Energy (keV)  Y-Axis: Counts
```

These commands define a second instrument model, input a background data file for dataset number 2, set the `FAKEIT` parameters for dataset number 2, and define a second source model. The command `FAKEIT 2` then creates simulated dataset number 2. Both simulated datasets are then plotted, and the second dataset is written to the data file `fakeit3.pha`.

4. Simulate a dataset, without an instrument model:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POW[modela]
sherpa> DATASPACE (1:1000:2)
sherpa> FAKEIT TIME = 33483.2
sherpa> FAKEIT BACKSCALE = 0.0441895
sherpa> SHOW FAKEIT
Fakeit exposure time: 33483.2 seconds.
Fakeit backscale: 0.0441895

sherpa> modela.gamma=2
sherpa> modela.ampl.min=0.000001
sherpa> modela.ampl=0.0003
sherpa> modela.ref=1
sherpa> FAKEIT
sherpa> LPLOT DATA
```

Instead of defining an instrument model, the `DATASPACE` command is used to define the appropriate grid over which to calculate the model values.

5. Fake a 2-D image:

```
sherpa> DATASPACE (1:256:1,1:256:1)
sherpa> PARAMPROMPT OFF
sherpa> INSTRUMENT = FPSF2D[p]
sherpa> p.file = psf.fits
sherpa> SOURCE = GAUSS2D[g]
sherpa> g.xpos = 128
sherpa> g.ypos = 128
sherpa> g.ampl = 100
sherpa> g.fwhm = 25
sherpa> FAKEIT
```

1.33 FEFFILE

Specifies the Fits Embedded Function (FEF) file whose contents will be displayed with **FEFPLOT**, Section 1.34.

```
sherpa> FEFFILE "<filename>[<virtual_file_syntax>]"
```

where <**virtual_file_syntax**> is an optional filtering and/or binning command argument. Note that, whenever <**virtual_file_syntax**> is specified, <**filename**><**virtual_file_syntax**> must be surrounded by quotes, " ".

See **FEFPLOT**, Section 1.34 for more information, and further examples.

Examples:

1. Read in a part of a FEF file using a <**virtual_file_syntax**> filter expression:

```
sherpa> FEFFILE "data/fef_response.fits[function][ccd_id=0,chipx>=1,chipx<=256,chipy>=1,chipy<=32]"
```

1.34 FEFPLOT

Plots an instrument response stored in a Fits Embedded Function (FEF) file that is read in via **FEFFILE**, Section 1.33.

```
sherpa> FEFPLOT {<photon_energy> | <photon_wavelength>}
```

where {<**photon_energy**> | <**photon_wavelength**>} must be in the same units that are used for photon energies in the FEF file (conventionally, keV or Angstroms).

Examples:

1. Read in part of a FEF file and plot the response as a function of counts-space energy, at photon energy 2.4 keV:

```
sherpa> FEFFILE "data/fef_response.fits[function][ccd_id=0,chipx>=1,chipx<=256,chipy>=1,chipy<=32]"
sherpa> FEFPLOT 2.4
```

The appearance of the plot may be subsequently altered using ChIPS commands such as:

```
sherpa> LIMITS X 0 3
sherpa> REDRAW
```

1.35 FIT — RUN

Initiates fitting (optimization). The command [B]RUN is equivalent.

```
sherpa> {[B]FIT | [B]RUN} [<dataset range> | ALLSETS] [ID]
```

`FIT` and `RUN` are used in initiate fitting of all appropriate source *and* background datasets (*i.e.*, those for which `SOURCE` [Section 1.77] and/or `BACKGROUND` [Section 1.9] model stacks have been defined), while `BFIT` and `BRUN` are used to initiate fitting of appropriate background datasets only.

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to fit all appropriate datasets. The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

In CIAO 3.1 the definition of INSTRUMENT BACK is required for fitting PHA data if either background file or background models have been defined. INSTRUMENT BACK is set automatically when the PHA data file is input to *Sherpa*, however it is deleted if the NEW background file is input for a given data set. Thus the new INSTRUMENT BACK has to be defined on the command line before starting a simultaneous fit with the new background file.

Fitting may also be initiated via the *Sherpa*/S-Lang module function `run_fit`. (This is equivalent to `FIT`; there is no module function equivalent of `BFIT` in CIAO 3.0.)

Examples:

1. Fit a source model to one dataset:

   ```
   sherpa> READ DATA 1 example1.dat
   sherpa> PARAMPROMPT OFF
   Model parameter prompting is off
   sherpa> POLY[modela]
   sherpa> SOURCE 1 = modela
   sherpa> THAW modela.2 modela.3
   sherpa> FIT
   ```

   The fourth command, `SOURCE 1 = modela`, defines the *Sherpa* model `POLY` as the source model to be used for fitting dataset number 1. The last command, `FIT`, fits source model number 1 to its corresponding dataset. For this example, the following command is equivalent to the last command:

   ```
   sherpa> FIT 1
   ```

2. Fit a source model to one dataset, when multiple source models have been defined:

   ```
   sherpa> ERASE ALL
   ```

```
sherpa> READ DATA 1 example1.dat
sherpa> POLY[modela]
sherpa> GAUSS[modelb]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modelb
sherpa> THAW modela.2 modela.3
sherpa> FIT 1
```

The last command fits source model number 1 to its corresponding dataset. Note that source model number 2 does not have a corresponding dataset, and so no fitting for this model will be performed; thus for this example, the following command is equivalent to the last command:

```
sherpa> FIT
```

3. Fit a source model to one dataset, when multiple source models have been defined and multiple datasets have been input:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> POLY[modela]
sherpa> GAUSS[modelb]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modelb
sherpa> THAW modela.2 modela.3
sherpa> FIT 1
```

The last command fits source model number 1 to its corresponding dataset. Note that source model number 2 is not fit to its corresponding dataset, because only source model number 1 is specified in the `FIT` command. In this example, the following commands could be issued to fit source model number 2 to its corresponding dataset:

```
sherpa> GUESS SOURCE 2
sherpa> FIT 2
```

Note that the `GUESS SOURCE 2` command is issued, in order to start the initial parameter values and ranges, of source model number 2, at estimates based on the input dataset number 2. See the `GUESS` command for futher information.

4. Independently fit the same model shape to different datasets:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> POLY[modela]
sherpa> POLY[modelb]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modelb
sherpa> THAW modela.2 modela.3
sherpa> THAW modelb.2 modelb.3
sherpa> GUESS SOURCE 2
sherpa> FIT
```

The last command fits source model number 1 to its corresponding dataset (dataset number 1), and fits source model number 2 to its corresponding dataset (dataset number 2). In this example, source models 1 and 2 are the same model, but with separate and independent parameters. The `GUESS SOURCE 2` command is issued in order to start the initial parameter values and ranges, of source model number 2, at estimates based on the input dataset number 2. See the `GUESS` command for futher information. For this example, the following commands are equivalent to the last command:

```
sherpa> FIT 1,2
sherpa> FIT 1:2
sherpa> FIT ALLSETS
```

5. Perform a joint fit of two datasets using the same source model:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> POLY[modela]
sherpa> SOURCE 1:2 = modela
sherpa> THAW modela.2 modela.3
sherpa> FIT
```

The last command fits source model number 1 to its corresponding dataset (dataset number 1), and fits source model number 2 to its corresponding dataset (dataset number 2). In this example, source models 1 and 2 are the same model, with shared parameters. For this example, the following commands are equivalent to the last command:

```
sherpa> FIT 1,2
sherpa> FIT 1:2
sherpa> FIT ALLSETS
```

Note that in this example, the command `FIT 1` would have fit the source model only to dataset number 1; the command `FIT 2` would have fit the source model only to dataset number 2.

6. Independently fit the same model shape to different datasets, but link one source model parameter to another:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> POLY[modela]
sherpa> POLY[modelb]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modelb
sherpa> modelb.c0 => modela.c0
sherpa> THAW modela.2 modela.3
sherpa> FIT
```

The last command fits source model number 1 to its corresponding dataset (dataset number 1), and fits source model number 2 to its corresponding dataset (dataset number 2). In this example, source models 1 and 2 are the same model, but with separate and independent parameters, for all parameters except c0. Parameter c0 is a shared parameter, as set by the parameter expression, in the eighth command (see the `CREATE` (Section 1.23) command for further information on model parameter expressions). For this example, the following commands are equivalent to the last command:

```
sherpa> FIT 1,2
sherpa> FIT 1:2
sherpa> FIT ALLSETS
```

7. Perform a joint fit of some (but not all) input datasets using the same source model:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> READ DATA 3 example3.dat
sherpa> READ DATA 4 example4.dat
sherpa> POLY[modela]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modela
sherpa> SOURCE 3 = modela
sherpa> SOURCE 4 = modela
sherpa> THAW modela.2 modela.3
sherpa> FIT 2,3
```

The last command fits source model number 2 to its corresponding dataset (dataset number 2), and fits source model number 3 to its corresponding dataset (dataset number 3). In this example, source models 2 and 3 are the same model, with shared parameters. Note that because source models 1 through 4 are all the same model, with shared parameters, one may specify to simultaneously fit to any of the input datasets. The command

```
sherpa> FIT 1,4
```

fits the source model to datasets number 1 and 4. The command

```
sherpa> FIT 2:4
```

(or `FIT 2,3,4`) fits the source model to datasets number 2, 3, and 4. The command

```
sherpa> FIT 1:4
```

(or `FIT 1,2,3,4`, or `FIT`, or `FIT ALLSETS`) fits the source model to all of the input datasets.

8. Perform a fit to a background dataset:

```
sherpa> READ DATA 1 example1.dat
sherpa> READ BACK 1 example1_back.dat
sherpa> SOURCE 1 = POLY[modela]
sherpa> BACKGROUND 1 = POLY[modelb]
sherpa> BFIT 1
```

The command `BFIT` instructs *Sherpa* to fit `modelb` to the input background data only, ignoring the source data and source model stack. For this example, the following commands are equivalent to the last command:

```
sherpa> BFIT
```

or

```
sherpa> SOURCE 1 =
sherpa> FIT 1        # or simply FIT
```

1.36 FLUX

Calculates the unconvolved photon flux for source or background datasets.

```
sherpa> [B][P]FLUX [# [ID]] [{(<value>) | (<min>:<max>) | (<region descriptor>)} ] \
[ {(<model component>) | (<model stack>)} ]
```

`FLUX` and `PFLUX` are equivalent commands for computing source fluxes, while `BFLUX` and `BPFLUX` are equivalent commands for computing background fluxes.

`#` specifies the dataset over which the model is evaluated. The `ID` modifier is used only for computing background fluxes, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The default is to compute photon fluxes for all appropriate datasets (*i.e.*, those for which source/background expressions have been defined). The flux may be computed at one energy/wavelength, over a range of energies/wavelengths, or within a 2-D region, with the default being to compute the flux the total available range.

The flux may also be computed for individual model components, or for previously defined model stacks, with the default being to compute the flux using all model components in the `SOURCE` or `BACKGROUND` expression.

Note that `FLUX` computes only photon fluxes in CIAO 3.0; to compute energy fluxes, use `EFLUX` (Section 1.27), while to compute summations of observed or model counts, use `DCOUNTS` (Section 8.4) and `MCOUNTS` (Section 1.52), respectively.

A source or background model stack must be defined before a respective flux can be computed; see the `SOURCE`, Section 1.77, and `BACKGROUND`, Section 1.9, commands. This is true even if one computes the flux of an individual model component or of models defined in a model stack. (This limitation will be removed in a future version of *Sherpa*.)

For 1-D data, if:

- one energy (or wavelength) <`value`> is given, then the photon flux at that energy (or wavelength) is returned in units of photons/cm^2/sec/keV (or photons/cm^2/sec/Å).
- an energy or wavelength range is given with the <`min`> and <`max`> arguments, then the total integrated photon flux over the given range is returned in units of photons/cm^2/sec.
- no value or range is specified, then the total integrated photon flux over the full range of the dataset is returned in units of photons/cm^2/sec.

For 1-D data, if an instrument model is not used, the units are (perhaps incorrectly) assumed to be counts, or counts per bin-width. See the *Note on Units* below.

For 2-D data, if a region descriptor is given, then the total integrated photon flux within that region is returned; otherwise, the integration is carried out over the entire input image.

Note on Units: in its current incarnation, *Sherpa* has no explicit knowledge of data or model units. The units displayed with computed fluxes are defaults, generally correct for standard analyses of 1-D PHA energy/wavelength spectra (*XSPEC*-like analyses). They may be incorrect for non-standard analyses, or for analyses of 2-D spatial images with exposure maps, *etc.* The correct units can be determined by working backwards from the data, taking into account the exposure time, the units of the instrument model, the bin units, *etc.*

Tip: To perform background subtraction in *Sherpa*, the command `SUBTRACT`, Section 1.80, must be issued; this is in contrast to *XSPEC*, which performs background subtraction automatically.

The photon flux may be computed using the *Sherpa*/S-Lang module functions `get_flux` (or `get_pflux`) and `get_bflux` (or `get_bpflux`)).

Examples:

1. Calculate the integrated photon flux over the full energy range:

```
sherpa> FLUX
Flux for source dataset 1: 0.00579108 photons/cm**2/s
```

2. Calculate the integrated background photon flux over the range 2 to 10 keV:

```
sherpa> BPFLUX (2.0:10.0)
Flux for background dataset 1: 5.09639e-05 photons/cm**2/s
```

3. Calculate the photon flux at a single energy (2.0 keV) for the power-law component of a source expression:

```
sherpa> SOURCE 1 = XSWABS[A] * POW[P]
sherpa> FLUX 1 (2.0) P
Flux for source dataset 1: 0.000494404 photons/cm**2/s/keV
```

4. Calculate the total photon flux over the energy range 2.0 to 4.0 keV for a model stack:

```
sherpa> FOO = POW[P] + GAUSS[G]
sherpa> SOURCE 1 = XSWABS[A] * FOO
sherpa> PFLUX 1 (2:4) FOO
Flux for source dataset 1: 0.000498425 photons/cm**2/s
```

5. Calculate the total flux within circles of radius 5 and 10 in a 2-D image (note, the quotes are necessary):

```
sherpa> SOURCE = GAUSS2D[G]
sherpa> FLUX 1 "CIRCLE(247,257,5)"
Flux for source dataset 1: 1614.79 counts
sherpa> FLUX 1 "CIRCLE(247,257,10)"
Flux for source dataset 1: 3142.15 counts
```

1.37 FREEZE

Prohibits model parameter(s) from varying.

```
sherpa> FREEZE <arg_1> [<arg_2> ...]
```

<arg> may be:

Argument	Description
`<sherpa_modelname>.{<paramname> | <#>}`	Freezes the specified model component parameter.
`<modelname>.{<paramname> | <#>}`	Freezes the specified model component parameter.
`<sherpa_modelname>`	Freezes all parameters of the specified model component.
`<modelname>`	Freezes all parameters of the specified model component.
`<sherpa_model_stack> [# [ID]]`	Freezes the parameters of all model components within the specified model stack (`SOURCE`, `BACKGROUND`, `PILEUP`, [B]`NOISE`, or `KERNEL`). `#` is a dataset number (default 1); see `BACKGROUND` for an explanation of `ID`.
`<model_stack>`	Freezes the parameters of all model components within the user-defined model stack.

Table 1.13: `FREEZE` Command Arguments

The command `THAW` (Section 1.83) is used to allow model parameter values to vary.

In addition, model parameters may be frozen using the equivalent command

- `sherpa> <modelname>.{<paramname>|<#>}.TYPE = FREEZE`

If model parameter prompting is enabled, the user may freeze a parameter by entering `[<value>:<min>:<max>],-1` at the prompt; the -1 sets the parameter to be frozen.

Model parameters may also be frozen using the *Sherpa*/S-Lang module functions `set_frozen` and `set_par`.

Examples:

1. Freeze a model parameter:

   ```
   sherpa> PARAMPROMPT OFF
   ```

```
sherpa> GAUSS[modelb]
sherpa> FREEZE modelb.ampl
```

The last command freezes the parameter `ampl` of `modelb`.

2. Thaw a model parameter and freeze a model parameter:

```
sherpa> THAW modelb.3
sherpa> FREEZE modelb.3
```

The first command thaws the third parameter of `modelb`. The last command freezes the third parameter of `modelb`.

3. Thaw all model parameters and freeze two model parameters:

```
sherpa> THAW modelb
sherpa> FREEZE modelb.fwhm modelb.pos
```

First, all parameters of `modelb` are thawed. The last command then freezes the `fwhm` and `pos` parameters of `modelb`.

4. Freeze or thaw all source component parameters at once:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
sherpa> GAUSS[modelf]
sherpa> SOURCE 2 = modelc + modelf
sherpa> FREEZE SOURCE 2
sherpa> SHOW SOURCE 2
Source 2:

(modelc + modelf)
powlaw1d[modelc]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
 1  gamma frozen          1       -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl frozen          1      1e-20 3.4028e+38
gauss1d[modelf]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
 1   fwhm frozen         10 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl frozen          1-3.4028e+38 3.4028e+38
sherpa> THAW SOURCE 2
sherpa> SHOW SOURCE 2
(modelc + modelf)
powlaw1d[modelc]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
```

```
 1  gamma thawed          1        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1      1e-20 3.4028e+38
gauss1d[modelf]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed         10 1.1755e-38 3.4028e+38
 2    pos thawed          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
```

This example illustrates the use of `FREEZE SOURCE` and `THAW SOURCE` to freeze and thaw all source component parameters at once, respectively. Note that thawing of some model parameters (*e.g.*, `POWLAW1D.ref`) is not permitted.

5. Freeze a model parameter:

```
sherpa> modelb.ampl.TYPE = FREEZE
```

This command freezes the parameter `ampl` of `modelb`. The following commands are each equivalent:

```
sherpa> modelb.3.TYPE = FREEZE
sherpa> FREEZE modelb.ampl
sherpa> FREEZE modelb.3
```

6. Freeze model parameters:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> GAUSS[modelb]
modelb.fwhm parameter value [10] 2,-1
modelb.pos parameter value [0] ,-1
modelb.ampl parameter value [1]
sherpa> SHOW modelb
gauss1d[modelb]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen          2 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
```

In this example, parameters are frozen by entering <value>, -1 at the model parameter value prompt. Note that , -1 at the model parameter value prompt accepts the given initial parameter value and freezes that parameter.

7. Freeze all model parameters at once; thaw all source parameters at once:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> DATA data/example.pha
sherpa> GAUSS[modelc]
```

```
sherpa> SOURCE = modelb + modelc
sherpa> FREEZE modelc
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen          2 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen     0.7113     0.0071    71.1283
 2    pos frozen     0.9442     0.0276    14.5494
 3   ampl frozen     0.0001 1.0564e-06     0.0106
sherpa> THAW SOURCE
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed          2 1.1755e-38 3.4028e+38
 2    pos thawed          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     0.7113     0.0071    71.1283
 2    pos thawed     0.9442     0.0276    14.5494
 3   ampl thawed     0.0001 1.0564e-06     0.0106
```

Note that the command `FREEZE modelc` freezes all parameters of the source model component `modelc`, while `THAW SOURCE` thaws all parameters of both source model components.

1.38 FTEST

Computes significance using the F test.

```
sherpa> FTEST <dof_1> <stat_1> <dof_2> <stat_2>
```

The command arguments are:

Argument	Description
`<dof_1>`	Number of degrees of freedom (dof) for the fit of the null hypothesis (the simple model).
`<stat_1>`	Best-fit statistic for the null hypothesis.

<dof_2>	Number of degrees of freedom in the fit of the alternative hypothesis (more complex model).
<stat_2>	Best-fit statistic for the alternative hypothesis.

Table 1.14: Valid **FTEST** Command Arguments

The F-test is a model comparison test. Model comparison tests are used to select from two competing models that which best describes a particular dataset.

A model comparison test statistic T is created from the best-fit statistics of each fit; as with all statistics, it is sampled from a probability distribution $p(T)$. The test significance is defined as the integral of $p(T)$ from the observed value of T to infinity. The significance quantifies the probability that one would select the more complex model when in fact the null hypothesis is correct. A standard threshold for selecting the more complex model is significance < 0.05 (the "95% criterion" of statistics).

The F-test may be used if:

- the simpler of the two models is nested within the other, *i.e.*, one can obtain the simpler model by setting the extra parameters of the more complex model to default values, often zero or one;
- those normal distributions are not truncated by parameter space boundaries;
- and the best-fit statistics for each fit individually are sampled from the chi-square distribution.

If these conditions are fulfilled, then the observed F statistic is sampled from the F distribution, whose shape is a function of <dof_1> and <dof_2>. (The tail integral may be computed analytically using an incomplete beta function; see any basic statistics text for details.) If these conditions are not fulfilled, then the F-test significance may not be accurate.

One can create three F statistics out of the best-fit statistics for two model fits; the most powerful F-test, at least for line detection, uses the change in statistic from one fit to the next as <stat_1>, the additional number of parameters in the more complex model (<delta_dof>) for <dof_1>, the best-fit of the more complex model for <stat_2>, and the number of degrees of freedom for the more complex model for <dof_2>.

The F-test significance can also be retrieved using the *Sherpa*/S-Lang module function `get_ftest`.

Examples:

1. Compute the F-test significance given two fits:

   ```
   sherpa> FTEST 2 20.28 34 33.63
   significance = 0.000328079
   ```

   As noted above, for line fits, the first two numbers should be the difference in degrees of freedom and fit statistic, respectively, between the null hypothesis (continuum) fit (for which $\chi^2 = 53.91$ for 36 dof), and the alternative hypothesis (continuum-plus-line) fit (for which $\chi^2 = 33.63$ for 34 dof). In general, if the significance is smaller than a predefined threshold (*e.g.*, 0.05), then the more complex model is selected (as it is in this case). Otherwise the null hypothesis is selected.

1.39 GETX

Assigns x-axis values taken from a plot to model parameters.

```
sherpa> GETX [<arg>] <modelname>.{<paramname> | <#>} \
[<modelname>.{<paramname> | <#>} ...]
```

where <**modelname**> is the name that has been given to a model component by the user.

Notice that either <**paramname**> or <**#**> may be used to specify the parameter to which the value will be assigned. When the value to be assigned is the non-negative value of the difference between two x-axis values, $< arg >$ is 2.

To take an x-axis value from a plot, and assign it to a model parameter:

- Plot the data (see the `LPLOT` command).
- Issue the command **GETX** <**modelname**>.{<**paramname**> | <**#**>}.
- Place the cursor on the plot, at the desired x-axis value for the parameter, and click once with the left mouse button. Alternatively, hit the character e, at the desired x-axis value for the parameter.

To take two x-axis values from a plot, and assign the non-negative value of their difference to a model parameter:

- Plot the data (see the `LPLOT` command).
- Issue the command **GETX 2** <**modelname**>.{<**paramname**> | <**#**>}.
- Place the cursor on the plot, at the first desired x-axis value, and click once with the left mouse button. Place the cursor at the next desired x-axis value, and left-mouse-click again. Note that the character `e` may be used as an alternative to clicking the left mouse button.

See the `CREATE` command for further information and examples of alternative methods for setting model parameter values individually.

One may quit `GETX` by typing `q`. The previously assigned parameter value will be restored. Note, however, that if multiple parameter values are being set, only the one being altered when q is typed will have its value restored.

Examples:

1. Set a model parameter value to an x-axis value taken from a plot:

```
sherpa> DATA example.dat
```

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GAUSS[modelb]
sherpa> LPLOT DATA
sherpa> GETX modelb.pos
<left-mouse-click once at the desired x-axis value>
sherpa> SHOW modelb
```

The command `GAUSS`[`modelb`] assigns the name `modelb` to the *Sherpa* model component `GAUSS`. Next, the 1-D data are plotted. The command `GETX modelb.pos` readies the cursor for selecting the desired x-axis value from the plot, for parameter `pos` of `modelb`. The user should then left-mouse-click (or hit the character `e`) at the desired x-axis location. The command `SHOW modelb` will show the current parameter values and their ranges, for component `modelb`.

2. Set a model parameter value to the non-negative difference between two x-axis values taken from a plot:

```
sherpa> GETX 2 modelb.fwhm
<left-mouse-click once at the desired first x-axis value>
<and then left-mouse-click again at the desired second x-axis value>
sherpa> SHOW modelb
```

Here, the command `GETX 2 modelb.fwhm` readies the cursor for selecting the two desired x-axis values from the plot. The non-negative difference between these two values will be assigned to parameter fwhm of modelb. The command `SHOW modelb` will show the current parameter values and their ranges, for `modelb`.

1.40 GETY

Assigns y-axis values taken from a plot to model parameters.

```
sherpa> GETY [<arg>] <modelname>.{<paramname> | <#>} \
[<modelname>.{<paramname> | <#>} ...]
```

where <`modelname`> is the name that has been given to a model component by the user.

Notice that either <`paramname`> or <`#`> may be used to specify the parameter to which the value will be assigned. When the value to be assigned is the non-negative value of the difference between two y-axis values, $< arg >$ is 2.

To take a y-axis value from a plot, and assign it to a model parameter:

- Plot the data (see the `LPLOT` command).
- Issue the command `GETY` <`modelname`>`.`{<`paramname`> | <`#`>}.

- Place the cursor on the plot, at the desired y-axis value for the parameter, and click once with the left mouse button. Alternatively, hit the character `e`, at the desired y-axis value for the parameter.

To take two y-axis values from a plot, and assign the non-negative value of their difference to a model parameter:

- Plot the data (see the `LPLOT` command.)
- Issue the command `GETY 2 <modelname>.{<paramname> | <#>}`.
- Place the cursor on the plot, at the first desired y-axis value, and click once with the left mouse button. Place the cursor at the next desired y-axis value, and left-mouse-click again. Note that the character `e` may be used as an alternative to clicking the left mouse button.

See the `CREATE` command for further information and examples of alternative methods for setting model parameter values individually.

One may quit `GETY` by typing `q`. The previously assigned parameter value will be restored. Note, however, that if multiple parameter values are being set, only the one being altered when `q` is typed will have its value restored.

Examples:

1. Set a model parameter value to a y-axis value taken from a plot:

```
sherpa> DATA example.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GAUSS[modelb]
sherpa> LPLOT DATA
sherpa> GETY modelb.ampl
<left-mouse-click once at the desired y-axis value>
sherpa> SHOW modelb
```

   The command `GAUSS[modelb]` assigns the name modelb to the *Sherpa* model component `GAUSS`. Next, the 1-D data are plotted. The command `GETY modelb.ampl` readies the cursor for selecting the desired y-axis value from the plot, for parameter `ampl` of `modelb`. The user should then left-mouse-click (or hit the character `e`) at the desired y-axis location. The command `SHOW modelb` will show the current parameter values and their ranges, for component `modelb`.

2. Set a model parameter value to the non-negative difference between two y-axis values taken from a plot:

```
sherpa> GETY 2 modelb.ampl
<left-mouse-click once at the desired first y-axis value>
<and then left-mouse-click again at the desired second y-axis value>
sherpa> SHOW modelb
```

Here, the command `GETY 2 modelb.amply` readies the cursor for selecting the two desired y-axis values from the plot. The non-negative difference between these two values will be assigned to parameter `ampl` of `modelb`. The command `SHOW modelb` will show the current parameter values and their ranges, for `modelb`.

1.41 GOODNESS

Reports information on the goodness-of-fit.

`sherpa> GOODNESS [<dataset range> | ALLSETS]`

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to obtain information from all appropriate datasets.

`GOODNESS` reports to the user information about how well specified models fit to the data. At a minimum, it reports: the choice of statistic; the number of bins in the fit; the number of degrees of freedom (dof), *i.e.*, the number of bins minus the number of free parameters; and the statistic value. (See the documentation on the command `STATISTIC` for more information on how to set the current statistic within *Sherpa*.)

If the chosen statistic is one of the χ^2 statistics, or the `CSTAT` statistic, then more information is shown: the reduced statistic, *i.e.*, the statistic value divided by the number of dof; and the probability, or Q-value:

$$Q = \int_{\chi^2_{\rm obs}}^{\infty} d\chi^2 p(\chi^2|N-P)\,, \tag{1.3}$$

where $\chi^2_{\rm obs}$ represents a specific observed value of χ^2 (*e.g.*, resulting from a fit), $N-P$ is the number of degrees of freedom (number of bins minus number of free parameters), and $p(\chi^2|N-P)$ is the χ^2 probability sampling distribution.

Q measures the probability that one would observe the value $\chi^2_{\rm obs}$, or a larger value, if the assumed model is true and the best-fit model parameters are the true parameter values. A value that is too small (*e.g.*, $Q <$ 0.05) indicates that the selected model does not accurately portray the data, while a value that is too large (Q -< 1) indicates that the fit is “too good.” The usual cause of a fit that is too good is an overestimation of the errors (*e.g.*, by using `CHI GEHRELS` in the low-counts regime (see note below), or by adding in too much systematic error). Increasing the errors decreases $\chi^2_{\rm obs}$, and increases Q.

Note that the accuracy of Q is dependent upon whether the selected statistic is actually sampled from the χ^2 distribution! This may not be the case if the number of counts in any bin is too small (< 5-10).

The information output by `GOODNESS` may be retrieved using the *Sherpa*/S-Lang module function `get_goodness`.

Examples:

1. Report information on the goodness-of-fit:

```
sherpa> GOODNESS
Goodness: computed with Chi-Squared Data Variance

DataSet 1: 100 data points -- 99 degrees of freedom.
 Statistic value       = 82.7136
 Probability [Q-value] = 0.880939
 Reduced statistic     = 0.835491
```

1.42 GROUP

Causes *Sherpa* to apply a read-in bin grouping scheme to source or background data.

```
sherpa> [B]GROUP [# [ID]]
```

`GROUP` is used to group source data, while `BGROUP` is used to group background data.

`#` specifies the number of the dataset to which the grouping scheme is to be applied (default dataset number is 1). The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The commands `GROUP` and `UNGROUP` allow a user to toggle back and forth between the analysis of grouped and ungrouped data, after grouping assignments have been read into *Sherpa* via the command `READ GROUPS`. (In a future version of *Sherpa*, the `GROUP` may be issued automatically upon the reading in of groups.)

Note the issuing the `GROUP` causes *Sherpa* to delete any defined filters for the specified dataset.

Also note that even if the data are grouped, the user may continue to read in filters, weights, *etc.*, whose values map to the ungrouped data; *Sherpa* will do the grouping automatically:

- weights are averaged within groups;
- statistical and systematic errors are added in quadrature within groups;
- a grouped bin is noticed if any of its ungrouped component bins is noticed.

The user may also read in filters, weights, *etc.*, whose values map to the grouped data; these values are used directly.

NOTE: in CIAO 3.0, the commands `GROUP` and `UNGROUP` may not be used with PHA data that has a `GROUPING` column. This is because these data are grouped before *Sherpa* ever has control of them, and *Sherpa* thus has no knowledge of how the ungrouped data are distributed among bins. This will be changed in a future version of *Sherpa*.

Examples:

1. Input data from an ASCII file; input a grouping scheme; group and ungroup the data:

```
sherpa> ERASE ALL
sherpa> $more spec_short.dat
   1     59.0000       0       1       1       .05     0
   2     46.0000       0       1       1       .05     0
   3     49.0000       0       1       1       .05     5
   4     65.0000       0       -1      1       .05     0
   5     60.0000       2       -1      1       .05     0
   6     60.0000       2       1       .1      .05     0
   7     74.0000       2       -1      .1      .05     0
   8     58.0000       2       -1      .1      .05     0
   9     55.0000       2       1       .1      .05     0
   10    70.0000       1       -1      .1      .05     5
   11    61.0000       1       -1      .1      .05     0
   12    75.0000       1       1       1       .05     0
   13    56.0000       1       -1      1       .05     0
   14    60.0000       1       -1      1       .05     0
   15    45.0000       1       1       1       .05     0
   16    63.0000       1       -1      1       .05     5
   17    63.0000       1       -1      1       .05     0
   18    56.0000       0       1       1       .05     0
   19    58.0000       0       -1      1       .05     0
   20    54.0000       0       -1      1       .05     0
sherpa> DATA spec_short.dat
sherpa> READ GROUPS spec_short.dat 1 4
sherpa> GROUP
WARNING: any applied filters are being deleted!
sherpa> WRITE DATA
Write X-Axis: Bin  Y-Axis: Flux (Counts)
          1         59
          2         46
          3        174
          6        192
          9        186
         12        191
         15        171
         18        168
sherpa> UNGROUP
WARNING: any applied filters are being deleted!
```

1.43 GUESS

Estimates initial parameter values and ranges, based on input data.

```
sherpa> GUESS <arg> [# [ID]]
```

`#` specifies the dataset whose data are to be used to help estimated parameter values. The `ID` modifier is used only for guesses involving background data, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

<**arg**> may be:

Argument	**To perform an estimate for:**
`MODELS`	All current model components.
<**sherpa_model_stack**>	All current model components in the named stack (`SOURCE`, `BACKGROUND`, `PILEUP`, [B]`NOISE`, or `KERNEL`).
<**sherpa_modelname**>	The specified model component.
<**modelname**>	The specified model component.

Table 1.15: `GUESS` Command Arguments

In addition, <**arg**> may be either `ON` or `OFF`, to turn *Sherpa*'s parameter-value-guessing mechanism on and off respectively.

Note that:

- When a name is assigned to a model component, the model component is established, and if a dataset has been input then estimates are automatically made for the initial parameter values and their ranges (*i.e.*, the `GUESS` command is run without the user having to explicitly issue the command, unless the user has previous issued the command `GUESS OFF`, in which case the parameters take on default values).
- If multiple datasets have been input, and a model component has been established *without* using the `SOURCE` command, such as in

```
sherpa> GAUSS[g]
```

then parameter estimates will be made using the *first* dataset. To refine the estimates for, *e.g.*, dataset 3, do the following:

```
sherpa> SOURCE 3 = g
sherpa> GUESS SOURCE 3
```

- If multiple datasets have been input, and a model component has been established using the `SOURCE` command, such as in

```
sherpa> SOURCE 4 = GAUSS[g]
```

then parameter estimates will be made using the *numbered* dataset, in this example, dataset number 4.

- If a dataset has been input, and an RSP model defined, but INSTRUMENT has *not* been set, such as in

```
sherpa> RSP[a](example2.arf,example2.rmf)
sherpa> SOURCE = GAUSS[g]
```

then parameter estimates will be made using the *counts* data (divided by the observation time, if provided in the PHA header); *i.e.*, they will not take into account the area information provided in the ARF. For good parameter estimates, always set the instrument!

```
sherpa> RSP[a](example2.arf,example2.rmf)
sherpa> INSTRUMENT = a
sherpa> SOURCE = GAUSS[g]
```

- If a dataset has been input, and an RSP model defined with only an RMF (and no ARF), and the INSTRUMENT has been set, then the parameter estimates will be suboptimal, as they will not utilize effective area information that may have been included.
- The guessing algorithm may not be optimal for complex model expressions.

See the CREATE command for more information and other examples.

Examples:

1. Estimate initial parameter values and ranges, based on input dataset number 1:

   ```
   sherpa> PARAMPROMPT OFF
   sherpa> POLY[modela]
   sherpa> DATA example1.dat
   sherpa> GUESS modela
   ```

   This command estimates parameter values and ranges for component modela, based on dataset 1.

2. Estimate initial parameter values and ranges, based on a specified input dataset:

   ```
   sherpa> DATA 2 example2.dat
   sherpa> GUESS MODELS 2
   ```

   This command estimates parameter values and ranges for all current model components, based on dataset 2.

3. Estimate initial parameter values and ranges for source models and their corresponding datasets, when multiple source models have been defined and multiple datasets have been input:

   ```
   sherpa> ERASE ALL
   sherpa> READ DATA 1 example1.dat
   sherpa> READ DATA 2 example2.dat
   sherpa> READ DATA 3 example3.dat
   ```

```
sherpa> POW[modelc]
sherpa> GAUSS[modelf]
sherpa> BBODY[modelh]
sherpa> SOURCE 1 = modelc
sherpa> SOURCE 2 = modelf
sherpa> SOURCE 3 = modelh
sherpa> GUESS SOURCE 1
```

This command estimates initial parameter values and ranges, based on dataset number 1, for the components of the source model for dataset number 1 (*i.e.*, `modelc`). Note, however, that these estimates were already made automatically when component `modelc` was first established with the command `POW`[`modelc`].

```
sherpa> GUESS SOURCE 2
```

Estimates initial parameter values and ranges, based on dataset number 2, for the components of the source model for dataset number 2 (*i.e.*, `modelf`). Note that until this command is issued, the initial parameter values for `modelf` are estimates based on dataset number 1.

```
sherpa> GUESS SOURCE 3
```

Estimates initial parameter values and ranges, based on dataset number 3, for the components of the source model for dataset number 3 (*i.e.*, `modelh`). Note that until this command is issued, the initial parameter values for `modelh` are estimates based on dataset number 1.

1.44 IGNORE

Specifies a data portion to be excluded, for 1- or 2-D data.

```
sherpa> IGNORE [{SOURCE | BACK}] [<dataset range> | ALLSETS] [ID] <arg>
```

where <`dataset range`> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The modifiers `SOURCE` and `BACK` may be used to specify that the filter is to be applied to either the source or background data only. If neither is specified, then the filter is applied to *both* the source and background datasets. Note that in CIAO 3.1 when omitting the modifiers the definition of `INSTRUMENT BACK` is required for filtering PHA data in energy/wave if either background file or background models have been defined.

The `ID` modifier is used only for the command `IGNORE BACK` (see below), and even then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

<`arg`> is either:

Argument	**To exclude:**
`ALL`	The entire dataset.
`BAD`	All PHA channels with quality flags $>$ 0 (1-D only).
`FILTER` <**`filterExpr`**>	Bins with x-values satisfying <**`filterExpr`**>.
{`BINS` | `CHANNELS`} <**`filterExpr`**>	Bins whose sequential numbers satisfy <**`filterExpr`**>.
`ENERGY` <**`filterExpr`**>	Bins with x-values in energy space satisfying <**`filterExpr`**> (1-D only).
`WAVE` <**`filterExpr`**>	Bins with x-values in wavelength space satisfying <**`filterExpr`**> (1-D only).
`PLOT`	A range selected interactively from an open plot (1-D only).
`IMAGE`	A region selected interactively from an open image (2-D only).
{`IMAGE` | `LOGICAL`} <**`filterExpr`**>	Pixels with logical coordinates satisfying <**`filterExpr`**> (2-D only).
`PHYSICAL` <**`filterExpr`**>	Pixels with physical coordinates satisfying <**`filterExpr`**> (2-D only).
{`WCS` | `WORLD`} <**`filterExpr`**>	Pixels with world coordinates satisfying <**`filterExpr`**> (2-D only).

Table 1.16: `IGNORE` Command Arguments

The typical <**`filterExpr`**> for 1-D data is the *range* filter:

This Filter Expression:	**Means:**
`#:#`	Exclude all data from the first value to the second value, inclusive.
`#:`	Exclude all data above the given value, inclusive.
`:#`	Exclude all data below the given value, inclusive.
`#`	Exclude the data point corresponding to the given value.

Table 1.17: Specification of Range Filters for 1-D Data

For 2-D data, <filterExpr> is a quote-delimited region specification. See "ahelp dmregions" for more information on allowed region specifications.

For more information on filter expressions, see the *Sherpa Filtering* Chapter for the full <**`filterExpr`**> definition, as well as information regarding other filtering methods.

The difference between the arguments `FILTER` and {`BINS` | `CHANNELS`} may be illustrated with the following simplistic ASCII dataset:

```
x       y
-       -
22.6    10
22.7    9
```

```
22.8    12
22.9    7
```

The command `IGNORE FILTER 22.65:22.85` will cause the middle two bins to be filtered out. This is equivalent to `IGNORE BINS 2:3`, *i.e.*, ignore the 2nd and 3rd bins. For PHA datasets, `IGNORE FILTER` and `IGNORE BINS` will lead to the same result, since the x-values of the PHA channels are sequential integers: 1, 2, 3...

To use the commands `IGNORE ENERGY` or `IGNORE WAVE`, a PHA dataset must have been read in, and an `INSTRUMENT` model stack defined. Note that regardless of the current `ANALYSIS` setting, one may filter using either `IGNORE ENERGY` or `IGNORE WAVE`. See the examples below.

To specify regions to ignore within an image:

- Display the image and mark the filter region(s) (see the `IMAGE` command). *NOTE:* only `include` regions are allowed for filtering (as opposed to `exclude` regions). Instead of using `IGNORE IMAGE` with an `exclude` region, use `NOTICE IMAGE` with an `include` region instead.

  Verify results using `IMAGE FILTER`.
- Issue the command `IGNORE [<dataset range> | ALLSETS] IMAGE`. This will tell *Sherpa* to exclude data within the marked region(s).

To specify regions to ignore within a plot:

- Plot the data using the `LPLOT` command).
- Issue the command `IGNORE [<dataset range> | ALLSETS] PLOT`.
- Place the cursor on the plot, at the desired minimum x-axis value for the filter, and click once with the left mouse button (or hit the `e` key).
- Place the cursor at the desired maximum x-axis value, and click again with the left mouse button (or hit the `e` key). *Sherpa* will then exclude data within the marked region(s).

Ranges/regions to be ignored may be alternatively set using the *Sherpa*/S-Lang module functions `set_ignore`, `set_ignore_bad`, and `set_ignore2d`, *et al.*

Examples:

1. Exclude all data values:

   ```
   sherpa> DATA example.dat
   sherpa> IGNORE ALL
   ```

   The last command tells *Sherpa* to ignore all data in dataset 1.
2. Exclude a particular data range from the source, background, or both:

```
sherpa> DATA data/data.dat
sherpa> BACK data/data_bckg.dat
sherpa> IGNORE SOURCE FILTER 5:10
sherpa> LPLOT 2 DATA BACK
```

After the input of source and background data, the command `IGNORE SOURCE FILTER 5:10` tells *Sherpa* to exclude those bins in the *source* dataset whose x-axis values are equal to 5 through 10, inclusive. See the *Sherpa Filtering* Chapter for further examples of filter expressions and usage of the `IGNORE` command. With the final command, a plot of both the source and background data illustrates that the filter was applied to the source data only; the background data remains unfiltered.

```
sherpa> NOTICE ALL
sherpa> IGNORE BACK FILTER 5:10
sherpa> LPLOT 2 DATA BACK
sherpa> LPLOT BFILTER
```

The `NOTICE ALL` command eliminates any filters applied to the source and background datasets. The command `IGNORE BACK FILTER 5:10` tells *Sherpa* to exclude those bins in the *background* dataset whose x-axis values are equal to 5 through 10, inclusive. With the first `LPLOT` command, a plot of both the source and background data illustrates that the filter was applied to the background data only; the source data remains unfiltered. The final command plots the filter status of each background data point.

```
sherpa> NOTICE ALL
sherpa> IGNORE FILTER 5:10
sherpa> LPLOT 2 DATA BACK
sherpa> LPLOT FILTER
```

The `NOTICE ALL` command eliminates any filters applied to the source and background datasets. The command `IGNORE FILTER 5:10` tells *Sherpa* to exclude those bins in both the source and background datasets whose x-axis values are equal to 5 through 10, inclusive. With the first `LPLOT` command, a plot of both the source and background data illustrates that the filter was applied to both datasets. The final command plots the filter status of each source data point.

3. Exclude a 2-D region, specified from an image display:

```
sherpa> READ DATA example_img.fits
sherpa> IMAGE DATA
sherpa> NOTICE ALL
<mark include region(s)>
sherpa> IGNORE IMAGE
sherpa> IMAGE FILTER
```

In this example, 2-D image data is first displayed. Then the data filter is set to include the entire image, with the command `NOTICE ALL`. (This command is not needed in this particular example but is shown for completeness.) Next, the user chooses regions, by placing `include` markers on the displayed image. The command `IGNORE IMAGE` sets the data filter to exclude data within the marked regions. The command `IMAGE FILTER` displays the resulting filter.

4. Exclude a 1-D data range, specified from a plot display:

```
sherpa> READ DATA example.dat
sherpa> LPLOT DATA
sherpa> NOTICE ALL
sherpa> IGNORE PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER
sherpa> LPLOT DATA
```

In this example, 1-D data is first displayed. Then the data filter is set to include the entire dataset, with the command `NOTICE ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `IGNORE PLOT` readies the cursor for selecting the desired filter region, and the user should then left-mouse-click first at the desired minimum x-axis location and then again at the maximum x-axis location. After the second left-mouse-click, the *Sherpa* command prompt is returned. The command `LPLOT FILTER` will then plot the data region that you have marked for exclusion. Finally, the dataset is plotted again, with the command `LPLOT DATA`, showing that the selected regions have been properly excluded.

5. Exclude a 1-D data range, specified from a plot display, for dataset number 2:

```
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> LPLOT DATA 2
sherpa> NOTICE 2 ALL
sherpa> IGNORE 2 PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER 2
sherpa> LPLOT DATA 2
sherpa>
sherpa> NOTICE ALL
sherpa> IGNORE PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER
sherpa> LPLOT DATA
```

In this example, two 1-D datasets are input, and dataset number 2 is first displayed. Then, the data filter is set to include all of dataset number 2, with the command `NOTICE 2 ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `IGNORE 2 PLOT` readies the cursor for selecting the desired filter region for dataset number 2, and the user should then left-mouse-click first at the desired minimum x-axis location and then again at the maximum x-axis location. After the second left-mouse-click, the *Sherpa* command prompt is returned. The command `LPLOT FILTER 2` will then plot the data region that you have marked for exclusion from dataset number 2. Finally, dataset number 2 is plotted again, with the command `LPLOT DATA 2`, showing that the selected regions have been properly excluded.

Next, the user sets the data filter to include all of dataset number 1, with the command `NOTICE ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `IGNORE PLOT` readies the cursor for selecting the desired filter region for dataset number 1, and the user should then left-mouse-click first at the desired minimum x-axis location and then at the maximum x-axis location (from the plot that's currently displayed of dataset number 2). The command `LPLOT`

FILTER will then plot the data region that you have marked for exclusion from dataset number 1. Finally, dataset number 1 is plotted again, with the command LPLOT ALL, showing that the selected regions have been properly excluded. Note that dataset number 1 was interactively filtered, *from a display of dataset number 2*.

6. Exclude particular data values and ranges:

```
sherpa> DATA 3 example.dat
sherpa> NOTICE 3 ALL
sherpa> IGNORE 3 FILTER 4, 8:, 1:3
```

The first command, NOTICE 3 ALL, sets all of dataset number 3 to be included, and so clears any previous filters. (This command is not needed in this particular example but is shown for completeness.) The second command uses a filter expression to exclude those x-axis data values that are equal to 4, greater-than or equal to 8, or equal to 1 through 3, inclusive. See the *Sherpa Filtering* Chapter for further examples of filter expressions and usage of the IGNORE command.

7. Exclude PHA channels with quality flags different than 0:

```
sherpa> DATA data/example.pha
sherpa> IGNORE BAD
```

8. Filter PHA data by energy, or wavelength:

```
sherpa> ERASE ALL
sherpa> DATA data/example2.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
<directory_path>/example2.rmf
ARF is being input from:
<directory_path>/example2.arf
Background data are being input from:
<directory_path>/example2_bkg.pha
sherpa> ANALYSIS ENERGY
sherpa> NOTICE ALL
sherpa> IGNORE ENERGY 5:10
sherpa> LPLOT DATA
sherpa> NOTICE ALL
sherpa> IGNORE WAVE 0:20
sherpa> ANALYSIS WAVE
sherpa> LPLOT DATA
```

In this example, the dataset is filtered using IGNORE WAVE even though the ANALYSIS setting is ENERGY. This is possible because header information in the data file allowed for an instrument model to be automatically defined when the data were initially read.

9. Exclude a 2-D region specified from the command line, in physical coordinates:

```
sherpa> DATA data/example_img2.fits
sherpa> IGNORE PHYSICAL "CIRCLE(4010,3928,100)"
sherpa> IMAGE FILTER
```

To filter in physical coordinates requires no action on the part of the user beyond specifying the `PHYSICAL` modifier; *Sherpa* automatically performs the image-to-physical coordinate conversion (if it can). Note that the quote marks are required in CIAO 3.0 when specifying 2-D filter regions. (They are not required when specifying 1-D regions, as seen in the other examples above.) The command `IMAGE FILTER` displays the resulting filter.

1.45 IMAGE

Causes the specified 2-D data to be displayed, via *ds9*.

```
sherpa> IMAGE <arg> [# [ID]]
```

`#` specifies the number of the dataset (default dataset number is 1). The `ID` modifier is used for displaying background datasets, and then if and only if the *Sherpa* state object variable **`multiback`** is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The argument <**`arg`**> may be any of the following:

<table>
<tr><th>Argument</th><th>Displays</th></tr>
<tr><td>{{DATA | DCOUNTS} | {BACK | BDCOUNTS}}</td><td>The source|background data values</td></tr>
<tr><td>{ERRORS | BERRORS}</td><td>The estimated total errors for the source|background data values</td></tr>
<tr><td>{SYSERRORS | BSYSERRORS}</td><td>The assigned systematic errors for the source|background data values</td></tr>
<tr><td>{STATERRORS | BSTATERRORS}</td><td>The estimated statistical errors for the source|background data values</td></tr>
<tr><td>{{MODEL | MCOUNTS} | {BMODEL | BMCOUNTS}}</td><td>The (convolved) source|background model amplitudes</td></tr>
<tr><td>{FIT | BFIT}</td><td>The data, model, and absolute residuals for source|background, displayed in three frames</td></tr>
<tr><td>{DELCHI | BDELCHI}</td><td>The sigma residuals of the source|background fit: (data - model)/error</td></tr>
<tr><td>{RESIDUALS | BRESIDUALS}</td><td>The absolute residuals of the source|background fit: (data - model)</td></tr>
<tr><td>{RATIO | BRATIO}</td><td>The ratio (data/model) for source|background</td></tr>
<tr><td>{CHI SQU | BCHISQU}</td><td>The contributions to the χ^2 statistic from each source|background data point</td></tr>
<tr><td>{STATISTIC | BSTATISTIC}</td><td>The contributions to the current statistic from each source|background data point</td></tr>
<tr><td>{WEIGHT | BWEIGHT}</td><td>The statistic weight value assigned to each source|background data point</td></tr>
<tr><td>{FILTER | BFILTER}</td><td>The mask value (0|1) for each source|background data point</td></tr>
</table>

<sherpa_model_stack>	The (unconvolved) model amplitudes for the specified model stack (SOURCE, {BACKGROUND|BG}, or [B]NOISE)
<model_stack>	The (unconvolved) model amplitudes for the specified user-defined model stack
<sherpa_modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, GAUSS2D)
<modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, g)
{EXPMAP | BEXPMAP}	The unfiltered source|background 2-D EXPMAP file contents
{PSF | BPSF}	The unfiltered source|background 2-D PSF file contents

Table 1.18: IMAGE Command Arguments

If there is no open image window when an IMAGE command is given, one will be created automatically.

In CIAO 3.0, if one displays an image after filtering, what is displayed is the <data to image> * filter", shown over the range of the unfiltered dataset. In a future version of *Sherpa*, include/exclude regions may also be superimposed upon the image.

Also note that in CIAO 3.0, images are automatically resized to fit entirely within the image frame. The user may of course subsequently zoom in and/or out.

Related commands include IGNORE IMAGE (Section 1.44) and NOTICE IMAGE (Section 1.55), which are used to interactively define filter regions from an image display.

See the display chapter (Chapter 2) for more information regarding data display capabilities, including modifying various image characteristics.

1.45.1 Using Data Model Filters

It IS NOT possible to pass a Data Model virtual file specification to this command. While you can read in a large file with

```
sherpa> data "image.fits[opt mem=1000]"
```

the following DOES NOT work:

```
sherpa> image "model[opt mem=1000]"
```

This is because “model” is a token that the *Sherpa* parser interprets, and it will treat the rest of the string (“[opt mem=1000]”) as an error. A workaround is to write out the file with the “write” command:

```
sherpa> write model "out.fits[opt mem=1000]"
```

Then display out.fits in ds9.

Examples:

1. Display 2-D data:

   ```
   sherpa> DATA example_img.fits
   sherpa> IMAGE DATA
   ```

   The last command displays dataset number 1. Dataset number 1 must be a 2-D dataset.

2. Exclude a 2-D region; display the filtered region:

   ```
   sherpa> IGNORE FILTER "BOX(250,250,50,50)"
   sherpa> IMAGE FILTER
   ```

   The first command, `IGNORE FILTER ‘‘BOX(250,250,50,50)"` defines a filter for dataset number 1. The second command displays the defined filter region.

3. Display 2-D absolute residuals:

   ```
   sherpa> PARAMPROMPT OFF
   sherpa> SOURCE = GAUSS2D[g]
   sherpa> FIT
   sherpa> IMAGE RESIDUALS
   ```

1.46 INSTRUMENT — RESPONSE

Defines an expression to be used for modeling the instrument in source or background data analysis. The command RESPONSE is equivalent.

```
sherpa> {INSTRUMENT | RESPONSE} [{SOURCE | BACK}] [# [ID]] = <modelExpr>
```

where `#` may specify the number of the dataset (default dataset number is 1). The modifiers `SOURCE` and `BACK` may be used to specify the application of the instrument model stack to either the source or background data only. If neither is specified, then the model is used for *both* the source and background data. The `ID` modifier is used only for the command `INSTRUMENT BACK` (see below), and even then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The model expression, <`modelExpr`>, is an algebraic combination of one or more of the following elements:

```
{<sherpa_modelname> | <sherpa_modelname>[modelname] |
 <modelname> }
```

along with numerical values. The following operators are recognized: + * ( ); however, the operators + and * do not have the same meaning that they do when combining source model components. See below for details. (See the `CREATE` (Section 1.23) command for more information on establishing model components.)

Note that:

- In CIAO 3.1 the definition of INSTRUMENT BACK is required for both filtering and fitting the data if either background file or background models have been defined. INSTRUMENT BACK is set automatically when the PHA source file is input to *Sherpa*, however it is deleted if the NEW background file is input for a given data set, thus the new INSTRUMENT BACK has to defined on the command line before filtering and fitting the data with the new background file.
- By default, if the instrument model expression includes a component that has not previously been established, *Sherpa* will prompt for the initial parameter values for that component. This prompting can be turned off using the `PARAMPROMPT OFF` (Section 1.58) command.
- For the specific case of PHA data, a `RSP` instrument model (Section 5.51) is automatically defined if the `RESPFILE` and/or `ANCRFILE` header keywords are present and point to existing RMF and ARF files. The instrument model stack is then set to include the `RSP` model.

To reset an instrument model stack, issue the command:

```
sherpa> {INSTRUMENT | RESPONSE} [<dataset range> | ALLSETS] =
```

Instrument models describe instrument characteristics, such as effective area, a detector's energy response, or a mirror's point-spread function. They are convolved with, *e.g.*, a source model to compute the number of detected counts in each detector bin.

Instrument model stacks are thus fundamentally different from other model stacks in that the models they contain are not evaluated themselves, but are used to tranform (*i.e.*, fold) amplitude arrays ($y' = f(y, x)$, instead of $y = f(x)$). (In *Sherpa*, there are two types of transformations: multiplication by an array, and multiplication by a redistribution matrix.) Thus, as noted above, the instrument stack operators + and * do not have the same meaning as their model stack counterparts.

Instrument models bound by the * operator collectively take a photon spectrum y and fold it to a counts spectrum y'. The order of the models does not matter so long as only there is only one redistibutive model (*e.g.*, RMF or PSF) in the set.

```
sherpa> farf1d[a](arf.fits)
sherpa> frmf1d[r](rmf.fits)
sherpa> instrument = a*r
```

Here, the photon spectrum y is multiplied by the ARF, then folded through the RMF. This instrument stack is equivalent to

```
sherpa> instrument = rsp[a](rmf.fits,arf.fits)
```

Sets of instrument models separated by the + operator each fold the same evaluated photon spectrum y, with the resulting group of counts spectra being summed.

```
sherpa> farf1d[a1](arf_order1.fits)
sherpa> farf1d[a2](arf_order2.fits)
sherpa> frmf1d[r1](rmf_order1.fits)
sherpa> frmf1d[r2](rmf_order2.fits)
sherpa> instrument = a1*r1 + a2*r2
```

Here, the photon spectrum y is folded through the combination `a1*r1` to produce counts spectrum $c1$; y is also folded through the combination `a2*r2` to produce counts spectrum $c2$. The overall counts spectrum is then $c1 + c2$.

The two rules governing instrument stacks are:

- Instrument models sets separated by + operators must all do similar evaluations, *e.g.*, `instrument = arf + rmf*arf` is *not* allowed.
- As mentioned above, instrument model sets separated by + operators (or or a single instrument model set) cannot include two or more redistributive models, *e.g.*, `instrument = rmf*rmf` is *not* allowed.

Note that if one wants to do more complex operations (*e.g.*, dividing one arf by another as part of the folding process), one can use S-Lang to do the preliminary dirty work (for this example, the array division; the new array can then be loaded into *Sherpa* as the "arf" via *Sherpa*/S-Lang module function `load_arf`).

For *Sherpa* version 3.0.2, support for "dummy" instruments and datasets has been added:

- If data have been input and the instrument stack contains only an ARF, a dummy RMF will be created that maps the ARF bins to the data bins, if possible.
- If data have not been input and the instrument stack contains only an ARF, both a dummy RMF and a dummy PHA dataset will be created. (This is useful when one simply wants, *e.g.*, to visualize models along an energy or wavelength grid, without having to define a `DATASPACE` (Section 1.25).)
- If data have been input that contain information about the energy or wavelength grid (such as is contained in, *e.g.*, the `BIN_LO` and `BIN_HI` columns of a *Chandra* grating data file), then both a dummy ARF and RMF are created.

One may always overwrite the dummy instruments if they are not appropriate.

Also for *Sherpa* version 3.0.2, checks have been added that may lead to the instrument stack being deleted if a subsequent `DATA` or `DATASPACE` command is issued, if it appears that the models in the stack are incompatible with the input data. In future versions of *Sherpa*, the instrument stack may be deleted automatically in such situations to avoid analysis problems (*i.e.*, one will always have to specify instrument stacks *after* inputting data).

Also note that there are several instrument-model-stack-related *Sherpa*/S-Lang module functions.

Examples:

1. Define an instrument model using specified input response files:

```
sherpa> INSTRUMENT 1 = RSP[instrumentA]
instrumentA.rmf parameter value [] example.rmf
instrumentA.arf parameter value [] example.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
```

2. Define an instrument model, inputting the response files individually:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> INSTRUMENT 1 = FARF[iarf]*FRMF[irmf]
sherpa> iarf.arf = example.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> irmf.rmf = example.rmf
```

3. Define an instrument model using specified input response files:

```
sherpa> ERASE ALL
sherpa> RSP[instrumentA](example.rmf, example.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT 1 = instrumentA
```

4. Define an instrument model using specified input response files, including an encircled-energy arf (EEARF):

```
sherpa> ERASE ALL
sherpa> FARF1D[ieearf](example.eearf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> RSP[instrumentA](example.rmf, example.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT 1 = ieearf*instrumentA
```

5. Define an instrument model using specified input response files, with a path:

```
sherpa> ERASE ALL
sherpa> INSTRUMENT 1 = RSP[instrumentA]("data/example.rmf",
        "data/example.arf")
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
```

In all of the above examples, since neither a SOURCE or BACK argument is specified, the same instrument model is established for both the source and background.

6. Define different source and background instrument models using specified input response files (RMF and ARF only):

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> INSTRUMENT SOURCE 1 = RSP[instrumentAsrc]
instrumentAsrc.rmf parameter value [] example.rmf
instrumentAsrc.arf parameter value [] example.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT BACK 1 = RSP[instrumentAbkg]
instrumentAbkg.rmf parameter value [] example_bkg.rmf
instrumentAbkg.arf parameter value [] example_bkg.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
```

7. Define different source and background instrument models using specified input response files (RMF and ARF only):

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> INSTRUMENT SOURCE 1 = RSP[instrumentAsrc]
sherpa> instrumentAsrc.rmf = example.rmf
sherpa> instrumentAsrc.arf = example.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT BACK 1 = RSP[instrumentAbkg]
sherpa> instrumentAbkg.rmf = example_bkg.rmf
sherpa> instrumentAbkg.arf = example_bkg.arf
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
```

8. Define different source and background instrument models using specified input response files (RMF and ARF only):

```
sherpa> ERASE ALL
sherpa> RSP[instrumentAsrc](example.rmf, example.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT SOURCE 1 = instrumentAsrc
sherpa> RSP[instrumentAbkg](example_bkg.rmf, example_bkg.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT BACK 1 = instrumentAbkg
```

9. Define different source and background instrument models using specified input response files (RMF and ARF only):

```
sherpa> ERASE ALL
sherpa> INSTRUMENT SOURCE 1 = RSP[instrumentAsrc]("data/example.rmf",
        "data/example.arf")
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT BACK 1 = RSP[instrumentAbkg]("data/example_bkg.rmf",
        "data/example_bkg.arf")
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
```

10. Automatically define instrument models using input response files specified in dataset header(s):

```
sherpa> ERASE ALL
sherpa> DATA data/example.pi
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: statistical errors specified in the PHA file.
         These are currently IGNORED.  To use them, type:
         READ ERRORS "<filename>[cols CHANNEL,STAT_ERR]" fitsbin
RMF is being input from:
  <directory_path>/example.rmf
ARF is being input from:
  <directory_path>/example.arf
Background data are being input from:
  <directory_path>/example_bkg.pi
WARNING: statistical errors specified in the PHA file.
         These are currently IGNORED.  To use them, type:
         READ ERRORS "<filename>[cols CHANNEL,STAT_ERR]" fitsbin
sherpa> SHOW
...
------------------------------
Defined analysis model stacks:
------------------------------

instrument source 1 = AutoReadResponse
instrument back 1 = AutoReadResponse

------------------------------------
Defined instrument model components:
------------------------------------

rsp1d[AutoReadResponse]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1    rmf string: "/data/simteste/Testing/sherpaTest/data/example.rmf"
 2    arf string: "/data/simteste/Testing/sherpaTest/data/example.arf"
```

In this example, the same instrument model was automatically defined for both the source and background data, since the input data file header referenced the response files named `example.rmf` and `example.arf`.

11. Define an instrument model using the point-spread function contained in the file psf.fits:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> FPSF2D[ps1]
ps1.file parameter value ["none"] psf.fits
ps1.xsize parameter value [32]
ps1.ysize parameter value [32]
ps1.xoff parameter value [0]
ps1.yoff parameter value [0]
ps1.fft parameter value [1]
sherpa> INSTRUMENT = ps1
```

The source model will be convolved with the PSF provided in the psf.fits file.

1.47 INTEGRATE

Controls the integration of model components.

```
sherpa> <modelname> INTEGRATE {ON | OFF}
```

where <**modelname**> is a name that has been given to a model component by the user.

Models are integrated over bins in energy-space or wavelength-space (before being folded through an instrument), or over bins in counts-space (if no instrument model is specified). If the `OFF` option is chosen then, for non-binned data the model values at the entered data points are used, while for binned data the model values at the left side of the bin are used in the fitting.

This command cannot be applied to *XSPEC* source models or to instrument models. Also, changing the integration status of a model component will have no affect if the data are not binned.

One can use the `SHOW` command to determine whether a model component is currently being integrated.

The user must be careful not to define an unacceptable mixture of additive/integrated and multiplicative/non-integrated model components, such as by adding two model components with different integration statuses. *Sherpa* will issue a warning when it detects such an unacceptable mixture.

See also the *Sherpa* Manual, Chapter “Sherpa Models”, Subsection “Source Model Integration”.

1.47.1 Why do the model units change when integrate is turned on/off?

In general, the relevant parameters of a model (*e.g.*, positions, radii, amplitudes) match up with the COORD setting. If you are working in image coordinates, model parameters will also refer to image coordinates. This can change, however, if you change the integrate setting of the model.

If you create a model, *e.g.* a beta2d model, and turn integration off (the default for beta2d), you will get the model value at a point (x,y). In that case, the units are just "counts" since the rest of the beta2d expression is unitless (see "ahelp beta2d" for details).

When you turn integrate on, you are telling Sherpa to bin up the model values for you. Sherpa evaluates the model at (x_lo, y_lo) and (x_hi, y_hi), and bins it so that you get all the counts that happened in that bin, not just at some particular point in data space. Since the bin size is being taken into account, the value really means "counts/bin" or "counts/pixel", not simply "counts".

To see the difference that integrate makes, use the "image model" command to display the model values in ds9. Change the integrate setting and image the model again to see how the values change.

Examples:

1. Turn off integration for a model component:

```
sherpa> POW[modelc]
modelc.gamma parameter value [0]
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
sherpa> modelc INTEGRATE OFF
```

2. Turn off integration for a power-law model component:

```
sherpa> POWLAW1D[p]
sherpa> SHOW p
powlaw1d[p]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1      1e-20 3.4028e+38
sherpa> p INTEGRATE OFF
sherpa> SHOW p
powlaw1d[p]  (integrate: off)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1      1e-20 3.4028e+38
```

1.48 INTERVAL-PROJECTION — INT-PROJ

Plots the fit statistic as a function of parameter value, using the `PROJECTION` algorithm. The commands `INT-PROJ` and `INTPROJ` are abbreviated equivalents.

```
sherpa> INTERVAL-PROJECTION [<dataset range> | ALLSETS] <arg>
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to create plots using data from all appropriate datasets.

The command-line argument may be:

Argument	**Description**
`<sherpa_modelname>.{<paramname> | <#>}`	A specified model component parameter (*e.g.*, `GAUSS.pos`).
`<modelname>.{<paramname> | <#>}`	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.19: `INTERVAL-PROJECTION` Command Argument

The user may configure `INTERVAL-PROJECTION` via the *Sherpa* state object structure `intproj`. The current values of the fields of this structure may be displayed using the command `print(sherpa.intproj)`, or using the more verbose *Sherpa*/S-Lang module function `list_intproj()`.

The structure fields are:

Field	**Description**
`fast`	If 1, use a fast optimization algorithm (`LEVENBERG-MARQUARDT` or `SIMPLEX`) regardless of the current `METHOD`. If 0, use the current `METHOD`.
`expfac`	A multiplicative factor that expands the grid limits estimated by the `COVARIANCE` algorithm, *if* the grid limits are determined automatically (see `arange`, and below).
`arange`	If 1, the grid limits are to be determined automatically. If 0, the grid limits are specified (see `min` and `max`).
`min`	Specifies the grid minimum. This is always a linear quantity, regardless of the setting of `log` (see below). The setting is ignored if `arange` = 1.
`max`	Specifies the grid maximum. This is always a linear quantity, regardless of the setting of `log` (see below). The setting is ignored if `arange` = 1.
`log`	Specifies whether to use a linear (0) or logarithmic (1) spacing of grid points.
`nloop`	Specifies the number of grid points.
`sigma`	Specifies the number of σ (*i.e.*, the change in statistic) for the plot.

Table 1.20: `intproj` Structure Fields

Field values may be set using directly, *e.g.*,

```
sherpa> sherpa.intproj.arange = 0
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_intproj()`.

The plot is created by varying each selected parameter's value on the determined (`arange = 1`) or specified (`arange = 0`) grid, and computing the best-fit statistic at each grid point. `INTERVAL-PROJECTION` differs from `INTERVAL-UNCERTAINTY` (Section 1.49) in that all other thawed parameters are allowed to float to new best-fit values, instead of being fixed to their best-fit values. This makes a plot created by `INTERVAL-PROJECTION` a more accurate rendering of the projected shape of statistical hypersurface, but but causes the computation to proceed more slowly. For a fuller theoretical description of error estimation, see `PROJECTION` (Section 1.61), `UNCERTAINTY` (Section 1.85), and `COVARIANCE` (Section 1.21).

If `arange = 1`, then the grid limits for the plot are determined automatically using the `PROJECTION` algorithm. The selected parameter's value is varied until the fit statistic is increased by ΔS, which is a function of `INTERVAL-PROJECTION.sigma` (*e.g.*, $\Delta S = 1$ if the statistic is χ^2 and `sigma = 1`).

The grid-point values and best-fit statistics at each grid point may be retrieved using the *Sherpa*/S-Lang module function `get_intproj`. See the examples below.

Examples:

1. List the current and default values of the `intproj` structure, and restore the default values:

```
sherpa> sherpa.intproj.arange = 0
sherpa> sherpa.intproj.log = 1
sherpa> sherpa.intproj.sigma = 5
sherpa> list_intproj()
Parameter   Current        Default                          Description
-----------------------------------------------------------------------
fast              1              1    Switch to LM/simplex: 0(n)/1(y)
expfac            3              3          Expansion factor for grid
arange            0              1              Auto-range: 0(n)/1(y)
min               0              0                      Minimum value
max               0              0                      Maximum value
log               1              0            Log-spacing: 0(n)/1(y)
nloop            20             20              Number of grid points
sigma             5              1                    Number of sigma
sherpa> restore_intproj()
sherpa> list_intproj()
Parameter   Current        Default                          Description
-----------------------------------------------------------------------
fast              1              1    Switch to LM/simplex: 0(n)/1(y)
expfac            3              3          Expansion factor for grid
arange            1              1              Auto-range: 0(n)/1(y)
min               0              0                      Minimum value
max               0              0                      Maximum value
log               0              0            Log-spacing: 0(n)/1(y)
nloop            20             20              Number of grid points
sigma             1              1                    Number of sigma
```

2. Plot χ^2 within the 3σ confidence interval for a fit:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -10
sherpa> FIT
...
sherpa> sherpa.intproj.sigma = 3
sherpa> INTERVAL-PROJECTION my.c0
Interval-Projection: computing grid size with covariance...done.
                     outer grid loop 20% done...
                     outer grid loop 40% done...
                     outer grid loop 60% done...
                     outer grid loop 80% done...
```

3. Plot χ^2 for the same fit within manually set grid limits:

```
sherpa> sherpa.intproj.arange = 0
sherpa> sherpa.intproj.min = -25
sherpa> sherpa.intproj.max = 25
sherpa> INTERVAL-PROJECTION my.c0
Interval-Projection: grid size set by user.
                     outer grid loop 20% done...
                     outer grid loop 40% done...
                     outer grid loop 60% done...
                     outer grid loop 80% done...
```

4. Save the results of `INTERVAL-PROJECTION` to an ASCII file:

```
[...run INTERVAL-PROJECTION...]
sherpa> my_var = get_intproj()
sherpa> writeascii("my_output.dat",my_var.x0,my_var.y)
sherpa> quit
Goodbye.
unix> more my_output.dat
-25     34.2524
-22.3684        27.5464
-19.7368        21.5764
...
```

1.48.1 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `INTERVAL-PROJECTION` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `INTERVAL-PROJECTION` may not be valid unless the parameters are at their best-fit values.

1.49 INTERVAL-UNCERTAINTY — INT-UNC

Plots the fit statistic as a function of parameter value, using the `UNCERTAINTY` algorithm. The commands `INT-UNC` and `INTUNC` are abbreviated equivalents.

```
sherpa> INTERVAL-UNCERTAINTY [<dataset range> | ALLSETS] <arg>
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to create plots using data from all appropriate datasets.

The command-line argument may be:

Argument	Description
<`sherpa_modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `GAUSS.pos`).
<`modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.21: `INTERVAL-PROJECTION` Command Argument

The user may configure `INTERVAL-UNCERTAINTY` via the *Sherpa* state object structure `intunc`. The current values of the fields of this structure may be displayed using the command `print(sherpa.intunc)`, or using the more verbose *Sherpa*/S-Lang module function `list_intunc()`.

The structure fields are:

Field	Description
`arange`	If 1, the grid limits are to be determined automatically. If 0, the grid limits are specified (see `min` and `max`).
`min`	Specifies the grid minimum. This is always a linear quantity, regardless of the setting of `log` (see below). The setting is ignored if `arange` = 1.
`max`	Specifies the grid maximum. This is always a linear quantity, regardless of the setting of `log` (see below). The setting is ignored if `arange` = 1.
`log`	Specifies whether to use a linear (0) or logarithmic (1) spacing of grid points.
`nloop`	Specifies the number of grid points.
`sigma`	Specifies the number of σ (*i.e.*, the change in statistic) for the plot.

Table 1.22: `regproj` Structure Fields

Field values may be set using directly, *e.g.*,

```
sherpa> sherpa.intunc.arange = 0
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_intunc()`.

The plot is created by varying each selected parameter's value on an automatically determined grid, and computing the best-fit statistic at each grid point. `INTERVAL-UNCERTAINTY` differs from `INTERVAL-PROJECTION` (Section 1.48) in that all other thawed parameters are fixed to their best-fit values, instead of being allowed to float to new best-fit values. This makes a plot created by `INTERVAL-UNCERTAINTY` a less accurate rendering of the projected shape of statistical hypersurface, but it can be faster to create. For a fuller theoretical description of error estimation, see `PROJECTION` (Section 1.61), `UNCERTAINTY` (Section 1.85), and `COVARIANCE` (Section 1.21).

The grid limits for the plot are determined automatically using the `UNCERTAINTY` algorithm. Each parameter's value is varied until the fit statistic is increased by ΔS, which is a function of `sigma` (*e.g.*, $\Delta S = 1$ if the statistic is χ^2 and `sigma = 1`).

The grid-point values and best-fit statistics at each grid point may be retrieved using the *Sherpa*/S-Lang module function `get_intunc`. See the examples below.

Examples:

1. List the current and default values of the `intunc` structure, and restore the default values:

```
sherpa> sherpa.intunc.arange = 0
sherpa> sherpa.intunc.log = 1
sherpa> sherpa.intunc.sigma = 5
sherpa> list_intunc()
Parameter   Current         Default                        Description
-----------------------------------------------------------------------
arange            0               1              Auto-range: 0(n)/1(y)
min               0               0                      Minimum value
max               0               0                      Maximum value
log               1               0             Log-spacing: 0(n)/1(y)
nloop           100             100              Number of grid points
sigma             5               1                    Number of sigma
sherpa> restore_intunc()
sherpa> list_intunc()
Parameter   Current         Default                        Description
-----------------------------------------------------------------------
arange            1               1              Auto-range: 0(n)/1(y)
min               0               0                      Minimum value
max               0               0                      Maximum value
log               0               0             Log-spacing: 0(n)/1(y)
nloop           100             100              Number of grid points
sigma             1               1                    Number of sigma
```

2. Plot χ^2 within the 3σ confidence interval for a fit:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
```

```
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -10
sherpa> FIT
...
sherpa> sherpa.intunc.sigma = 3
sherpa> INTERVAL-UNCERTAINTY my.c0
Interval-Uncertainty: computing grid size with uncertainty...done.
                      outer grid loop 20% done...
                      outer grid loop 40% done...
                      outer grid loop 60% done...
                      outer grid loop 80% done...
```

3. Save the results of `INTERVAL-UNCERTAINTY` to an ASCII file:

```
[...run INTERVAL-UNCERTAINTY...]
sherpa> my_var = get_intunc()
sherpa> writeascii("my_output.dat",my_var.x0,my_var.y)
sherpa> quit
Goodbye.
unix> more my_output.dat
-4.17797        9.25601
-4.08741        8.89599
-3.99684        8.54331
...
```

1.49.1 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `INTERVAL-UNCERTAINTY` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `INTERVAL-UNCERTAINTY` may not be valid unless the parameters are at their best-fit values.

1.50 JOURNAL

Turns on/off the writing of all commands to a file.

```
sherpa> JOURNAL {<filename> | OFF }
```

where <`filename`> is the name given to the ASCII file that is written. The command `JOURNAL OFF` terminates writing to the file. By default, writing of all commands to a file is turned off.

Examples:

1. Write all commands to a file:

   ```
   sherpa> JOURNAL tmpfil
   ```

   This command initiates writing a record of all commands to the file tmpfil.

2. Turn off writing of all commands to a file:

   ```
   sherpa> JOURNAL OFF
   ```

   This command terminates writing to the file `tmpfil`.

1.51 LPLOT

Causes the specified 1-D data to be displayed, via *ChIPS*.

```
sherpa> LPLOT [<num_plots>] <arg_1> [# [ID]] [<arg_2> [# [ID]] ...]
```

<`num_plots`> specifies the number of plotting windows to open within the *ChIPS* pane (default 1); that number sets the number of subsequent arguments. For each subsequent argument, `#` specifies the number of the dataset (default dataset number is 1), and the `ID` modifier is used for displaying background datasets, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The argument <`arg_n`> may be any of the following:

<table>
<tr><th>Argument</th><th>Displays</th></tr>
<tr><td>{{DATA | DCOUNTS} | {BACK | BDCOUNTS}}</td><td>The source|background data values (with errors)</td></tr>
<tr><td>{UDATA | UBACK}</td><td>The unconvolved source|background data values (with errors)</td></tr>
<tr><td>{ERRORS | BERRORS}</td><td>The estimated total errors for the source|background data values</td></tr>
<tr><td>{SYSERRORS | BSYSERRORS}</td><td>The assigned systematic errors for the source|background data values</td></tr>
<tr><td>{STATERRORS | BSTATERRORS}</td><td>The estimated statistical errors for the source|background data values</td></tr>
<tr><td>{{MODEL | MCOUNTS} | {BMODEL | BMCOUNTS}}</td><td>The (convolved) source|background model amplitudes</td></tr>
<tr><td>{FIT | BFIT}</td><td>The data (with errors) and model for source|background</td></tr>
<tr><td>{UFIT | BUFIT}</td><td>The unconvolved data (with errors) and source model for source|background</td></tr>
</table>

{DELCHI | BDELCHI}	The sigma residuals of the source|background fit: (data - model)/error
{RESIDUALS | BRESIDUALS}	The absolute residuals of the source|background fit: (data - model)
{RATIO | BRATIO}	The ratio (data/model) for source|background
{CHI SQU | BCHISQU}	The contributions to the χ^2 statistic from each source|background data point
{STATISTIC | BSTATISTIC}	The contributions to the current statistic from each source|background data point
{WEIGHT | BWEIGHT}	The statistic weight value assigned to each source|background data point
{FILTER | BFILTER}	The mask value (0|1) for each source|background data point
{GROUP | BGROUP}	The grouping value (1|-1) associated with each source|background data point
{QUALITY | BQUALITY}	The quality value (0|2|5) associated with each source|background data point
<sherpa_model_stack>	The (unconvolved) model amplitudes for the specified model stack (SOURCE, {BACKGROUND|BG}, [B]NOISE, or KERNEL)
<model_stack>	The (unconvolved) model amplitudes for the specified user-defined model stack
<sherpa_modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, GAUSS)
<modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, g)
{ARF | BARF}	The unfiltered source|background 1-D ARF file contents
{PSF | BPSF}	The unfiltered source|background 1-D PSF file contents

Table 1.23: LPLOT Command Arguments

If there is no open plotting window when an LPLOT command is given, one will be created automatically.

Related commands include:

- IGNORE PLOT (Section 1.44) and NOTICE PLOT (Section 1.55), which are used to interactively define filter regions from a plot display.
- GETX (Section 1.39) and GETY (Section 1.40), which are used to interactively define model component parameter values from a plot display.

The appearance of plots generated with this command can be changed by modifying the fields of certain state objects. See the ahelp for *Sherpa* or for sherpa.plot for more information.

NOTE: all *ChIPS* commands may be used from within *Sherpa* to modify plot characteristics. In order to view these changes, the `REDRAW` command must be issued.

See the display chapter (Chapter 2) for more information regarding data display capabilities, including modifying various plot characteristics.

Examples:

1. Display 1-D data:

   ```
   sherpa> DATA 3 example.dat
   sherpa> LPLOT DATA 3
   ```

   The last command plots dataset number 3. Dataset number 3 must be a 1-D dataset.

2. Exclude a data range and display the filtered range:

   ```
   sherpa> DATA example.dat
   sherpa> IGNORE ALL
   sherpa> NOTICE FILTER 2:70
   sherpa> LPLOT FILTER
   ```

   The command `IGNORE ALL` clears any previous filters, then the next command defines a filter for dataset number 1. The last command displays the defined filter.

3. Exclude a background data range and display the filtered background range:

   ```
   sherpa> DATA example.dat
   sherpa> BACK example_bkg.dat
   sherpa> IGNORE BACK ALL
   sherpa> NOTICE BACK FILTER 2:70
   sherpa> LP BFILTER
   ```

   The command `IGNORE BACK ALL` clears any previous background filters, and the next command defines a filter for the background of dataset number 1. The last command displays the defined background filter.

4. Have plotted data written to output data files:

   ```
   sherpa> ERASE ALL
   sherpa> DATA data/example1a.dat
   sherpa> sherpa.dataplot.x_errorbars = 0
   sherpa> sherpa.dataplot.y_errorbars = 0
   sherpa> LPLOT DATA
   sherpa> STORE myplot1.chp
   sherpa> SAVE ALL mysession1.shp
   sherpa> ls myplot1.chp*
   myplot1.chp
   myplot1.chp.fits
   sherpa> ls mysession1.shp*
   mysession1.shp
   sherpa> EXIT
   Goodbye.
   ```

The `STORE` command (a *ChIPS* command) specifies that a record of the commands used to generate the plot be saved in an ASCII file named myplot1.chp. The command `SAVE ALL mysession1.shp` is useful for saving the current state of the Sherpa session; it may then be restored at a later time.

This example also shows the use of a state object to modify the appearance of a plot (see the ahelp for sherpa.dataplot for more information).

5. Display 1-D data, the fit, and the fit residuals, using two windows:

```
sherpa> DATA example.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modela]
sherpa> SOURCE = modela
sherpa> FIT 1
sherpa> LPLOT 2 FIT RESIDUALS
```

This command displays a plot with the data and source model together in the first window, and a plot with the residuals of the fit in the second window.

6. Display 1-D data and fit, for two different datasets, using two windows:

```
sherpa> DATA 2 example2.dat
sherpa> LPLOT 2 DATA 1 DATA 2
```

This command displays a plot with dataset number 1 in the first window, and dataset number 2 in the second window. The following commands are all equivalent:

```
sherpa> LP 2 DATA 1 DATA 2
sherpa> LP 2 DATA 1:2
sherpa> LP 2 DATA ALL
```

7. Display 1-D fit and residuals for first dataset, and also display second dataset:

```
sherpa> LP 3 FIT 1 RESIDUALS 1 DATA 2
```

This command displays a plot with dataset number 1 and its fit in the first window, the residuals in the second window, and dataset number 2 in the third window.

8. Plot multiple datasets:

```
sherpa> ERASE ALL
sherpa> DATA 1 data/example1.dat
sherpa> DATA 2 data/example2.dat
sherpa> DATA 3 data/example3.dat
sherpa> DATA 4 data/example4.dat
sherpa> LPLOT 4 DATA ALL
```

This `LPLOT` command plots all of the datasets, each dataset in a window – dataset number 1 in the first window, dataset number 2 in the second window, *etc.* In this example, the following command produces the same result:

```
sherpa> LPLOT 4 DATA 1:4
```

1.52 MCOUNTS

Calculates the sum of convolved model amplitudes for source or background datasets.

```
sherpa> [B]MCOUNTS [# [ID]] [{(<value>) | (<min>:<max>) | (<region descriptor>)} ] \
[ {(<model component>) | (<model stack>)} ]
```

`MCOUNTS` is used for summing source model counts, while `BMCOUNTS` is used for summing background model counts.

`#` specifies the dataset over which the source model is evaluated. The `ID` modifier is used only for summing background counts, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The default is to do summations for all appropriate datasets (*i.e.*, those for which source/background expressions have been defined). The summation may be computed at one energy/wavelength, over a range of energies/wavelengths, or within a 2-D region, with the default being to compute the flux the total available range.

The summation may also be computed for individual model components, or for previously defined model stacks, with the default being to compute the summation using all model components in the `SOURCE` or `BACKGROUND` expression.

A source or background model stack must be defined before a respective flux can be computed; see the `SOURCE`, Section 1.77, and `BACKGROUND`, Section 1.9, commands. This is true even if one computes the flux of an individual model component or of models defined in a model stack. (This limitation will be removed in a future version of *Sherpa*.)

For 1-D data, if

- one energy (or wavelength or bin number) <value> is given, then the model counts is returned for the bin associated with that value.
- an energy or wavelength range is given with the <`min`> and <`max`> arguments, then the total summed model counts for the given range is returned.
- no value or range is specified, then the total summed model counts for the full range of the dataset is returned.

For 2-D data, if a region descriptor is given, then the summation of model counts within that region is returned; otherwise, the summation is carried out over the entire input image.

Tip: To perform background subtraction in *Sherpa*, the command `SUBTRACT`, Section 1.80, must be issued; this is in contrast to *XSPEC*, which performs background subtraction automatically.

The summation of model counts may be done using the *Sherpa*/S-Lang module functions `get_mcounts_sum` and `get_bmcounts_sum`.

Examples:

1. Calculate the summation of model counts over the full energy range:

```
sherpa> MCOUNTS
Model counts for source dataset 1: 1014.89 counts
```

2. Calculate the summation of background model counts over the range 2 to 10 keV. If the areas of the source and background extraction regions are different (as manifested by differences in, *e.g.*, the BACKSCAL keywords in the headers of the source and background data files), then the number of expected background counts in the source region will also be shown.

```
sherpa> BMCOUNTS (2.0:10.0)
Model counts for background dataset 1: 456.003 counts
      ...scaled to source region: 140.496 counts
```

3. Calculate the summation of model counts at a single energy (2.0 keV) for the power-law component of a source expression:

```
sherpa> SOURCE 1 = XSWABS[A] * POW[P]
sherpa> MCOUNTS 1 (2.0) P
Model counts for source dataset 1: 1.07005 counts
```

4. Calculate the summation of model counts over the energy range 2.0 to 4.0 keV for a model stack:

```
sherpa> FOO = POW[P] + GAUSS[G]
sherpa> SOURCE 1 = XSWABS[A] * FOO
sherpa> MCOUNTS 1 (2:4) FOO
Model counts for source dataset 1: 83.346 counts
```

5. Calculate the summation of model counts within circles of radius 5 and 10 in a 2-D image (note, the quotes are necessary):

```
sherpa> SOURCE = GAUSS2D[G]
sherpa> MCOUNTS 1 "CIRCLE(247,257,5)"
Flux for source dataset 1: 1614.79 counts
sherpa> MCOUNTS 1 "CIRCLE(247,257,10)"
Flux for source dataset 1: 3142.15 counts
```

1.53 METHOD — SEARCHMETHOD

Specifies the optimization method. The command SEARCHMETHOD is equivalent.

```
sherpa> {METHOD | SEARCHMETHOD} <sherpa_methodname>
```

The following optimization methods are featured in *Sherpa*:

<sherpa_methodname>	Description
`GRID`	A grid search of parameter space, with no optimization.
`GRID-POWELL`	A grid search of parameter space, with optimization done starting from each grid point, using the `Powell` method.
{`LEVENBERG-MARQUARDT` | `LEV-MAR` | `LM`}	Levenberg-Marquardt optimization.
`MONTECARLO`	A Monte Carlo search of parameter space, with no optimization.
`MONTE-LM`	A Monte Carlo search of parameter space, with optimization done starting from each randomly selected point, using the `LEVENBERG-MARQUARDT` method.
`MONTE-POWELL`	A Monte Carlo search of parameter space, with optimization done starting from each randomly selected point, using the `POWELL` method.
`POWELL`	Powell optimization.
{`SIGMA-REJECTION` | `SIG-REJ`}	Optimization combined with data cleansing: outliers are filtered from the data.
`SIMPLEX`	Simplex optimization.
`SIMUL-ANN-1`	A simulated annealing search, with one parameter varied at each step and no optimization.
`SIMUL-ANN-2`	A simulated annealing search, with all parameters varied at each step and no optimization.
`SIMUL-POW-1`	A combination of `SIMUL-ANN-1` with `POWELL` optimization.
`SIMUL-POW-2`	A combination of `SIMUL-ANN-2` with `POWELL` optimization.
`USERMETHOD`	A user-defined method of optimization.

Table 1.24: Description of *Sherpa*'s Optimization Methods

The default optimization method in CIAO 3.0 is `LEVENBERG-MARQUARDT`.

The current optimization method, and its parameter names and values, may be listed with the command `SHOW METHOD`. Values for these optimization parameters may be set individually using one of the following command syntax options:

```
sherpa> <sherpa_methodname>.<paramname> = <value>
sherpa> <sherpa_methodname>.<paramname>.VALUE = <value>
sherpa> <sherpa_methodname>.<#> = <value>
sherpa> <sherpa_methodname>.<#>.VALUE = <value>
```

where:

- {<**paramname**> | <**#**>}: Specifies the parameter whose value is to be set.
- <**value**>: The numerical (or string) value to which the parameter should be set.

Parameter listings for each optimization method are given in the *Sherpa Methods* chapter. (Alternatively, one may peruse the documentation for each <**sherpa_methodname**>.)

The user may access a string giving the name of the current optimization method via the *Sherpa*/S-Lang module function `get_method_expr`.

Examples:

1. Specify an optimization method to be used; set an optimization parameter value:

```
sherpa> METHOD POWELL
sherpa> SHOW METHOD
Optimization Method: Powell

      Name       Value         Min         Max                     Description
      ----       -----         ---         ---                     -----------
  1   iters       2000           1       10000    Maximum number of iterations
  2     eps      1e-06       1e-09       1e-03             Fractional accuracy
  3     tol      1e-06       1e-08         0.1             Tolerance in lnmnop
  4    huge      1e+10        1000       1e+12             Fractional accuracy

sherpa> POWELL.eps = .0009
sherpa> SHOW METHOD
Optimization Method: Powell

      Name       Value         Min         Max                     Description
      ----       -----         ---         ---                     -----------
  1   iters       2000           1       10000    Maximum number of iterations
  2     eps      9e-04       1e-09       1e-03             Fractional accuracy
  3     tol      1e-06       1e-08         0.1             Tolerance in lnmnop
  4    huge      1e+10        1000       1e+12             Fractional accuracy
```

2. Set an optimization parameter value:

```
sherpa> POWELL.1 = 100
sherpa> SHOW METHOD
Optimization Method: Powell

      Name       Value         Min         Max                     Description
      ----       -----         ---         ---                     -----------
  1   iters        100           1       10000    Maximum number of iterations
  2     eps      1e-06       1e-09       1e-03             Fractional accuracy
  3     tol      1e-06       1e-08         0.1             Tolerance in lnmnop
  4    huge      1e+10        1000       1e+12             Fractional accuracy
```

3. Set an optimization parameter value:

```
sherpa> POWELL.eps.VALUE = .0007
sherpa> SHOW METHOD
Optimization Method: Powell

      Name       Value          Min          Max                    Description
      ----       -----          ---          ---                    -----------
   1  iters       2000            1        10000    Maximum number of iterations
   2    eps      7e-04        1e-09        1e-03             Fractional accuracy
   3    tol      1e-06        1e-08          0.1             Tolerance in lnmnop
   4   huge      1e+10         1000        1e+12             Fractional accuracy
```

4. Set an optimization parameter value:

```
sherpa> POWELL.1.VALUE = 100
sherpa> SHOW METHOD
Optimization Method: Powell

      Name       Value          Min          Max                    Description
      ----       -----          ---          ---                    -----------
   1  iters        100            1        10000    Maximum number of iterations
   2    eps      1e-06        1e-09        1e-03             Fractional accuracy
   3    tol      1e-06        1e-08          0.1             Tolerance in lnmnop
   4   huge      1e+10         1000        1e+12             Fractional accuracy
```

1.54 MLR

Computes significance using the Maximum Likelihood Ratio test.

```
sherpa> MLR <delta_dof> <delta_stat>
```

The command arguments are:

Argument	**Description**
<delta_dof>	The difference in the number of degrees of freedom (dofs) between the fits of the null and alternative (more complex) hypotheses.
<delta_stat>	The difference in the best-fit statistics between the two fits.

Table 1.25: MLR Command Arguments

The Maximum Likelihood Ratio (MLR) test is a model comparison test. Model comparison tests are used to select from two competing models that which best describes a particular dataset. A model comparison test statistic T is created from the best-fit statistics of each fit; as with all statistics, it is sampled from a probability distribution $p(T)$. The test significance is defined as the integral of $p(T)$ from the observed value of T to infinity. The significance quantifies the probability that one would select the more complex model when in fact the null hypothesis is correct. A standard threshold for selecting the more complex model is significance < 0.05 (the "95% criterion" of statistics).

The MLR test may be used if:

- the simpler of the two models is nested within the other, *i.e.*, one can obtain the simpler model by setting the extra parameters of the more complex model to default values, often zero;
- the extra parameters have values sampled from normal distributions under the null hypothesis (*i.e.*, if one samples many datasets given the null hypothesis and fits these data with the more complex model, the distributions of values for the extra parameters must be Gaussian);
- those normal distributions are not truncated by parameter space boundaries; and
- the best-fit statistics for each fit individually are sampled from the χ^2 distribution.

If these conditions are fulfilled, then the change in statistic from one fit to the other (<`delta_stat`>) is sampled from the χ^2 distribution for <`delta_dof`> degrees of freedom. If these conditions are not fulfilled, then the MLR test significance may not be accurate.

The MLR test significance can also be retrieved using the *Sherpa*/S-Lang module function `get_mlr`.

Examples:

1. Perform the Maximum Likelihood Ratio test. Fit two models to the data, one with two more parameters than the other. The improvement in χ^2 (or the Cash statistic) is 20.0. Determine the significance by computing the tail integral of the χ^2 distribution for 2 degrees of freedom from 20.0 to infinity:

   ```
   sherpa> MLR 2 20.00
   significance = 4.53999e-05
   ```

   If this significance is smaller than the predefined threshold for accepting the more complex model (*e.g.*, 0.05), then the more complex model is selected. Otherwise, the null hypothesis is selected.

1.55 NOTICE

Specifies a data portion to be included, for 1- or 2-D data.

```
sherpa> NOTICE [{SOURCE | BACK}] [<dataset range> | ALLSETS] [ID] <arg>
```

where <`dataset range`> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The modifiers `SOURCE` and `BACK` may be used to specify that the filter is to be applied to either the source or background data only. If neither is specified, then the filter is applied to *both* the source and background datasets. Note that in CIAO 3.1 when omitting the modifiers the definition of `INSTRUMENT BACK` is required for filtering PHA data in energy/wave if either background file or background models have been defined.

The `ID` modifier is used only for the command `IGNORE BACK` (see below), and even then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

<`arg`> is either:

Argument	**To include:**
`ALL`	The entire dataset.
`FILTER` <`filterExpr`>	Bins with x-values satisfying <`filterExpr`>.
{`BINS` | `CHANNELS`} <`filterExpr`>	Bins whose sequential numbers satisfy <`filterExpr`>.
`ENERGY` <`filterExpr`>	Bins with x-values in energy space satisfying <`filterExpr`> (1-D only).
`WAVE` <`filterExpr`>	Bins with x-values in wavelength space satisfying <`filterExpr`> (1-D only).
`PLOT`	A range selected interactively from an open plot (1-D only).
`IMAGE`	A region selected interactively from an open image (2-D only).
{`IMAGE` | `LOGICAL`} <`filterExpr`>	Pixels with logical coordinates satisfying <`filterExpr`> (2-D only).
`PHYSICAL` <`filterExpr`>	Pixels with physical coordinates satisfying <`filterExpr`> (2-D only).
{`WCS` | `WORLD`} <`filterExpr`>	Pixels with world coordinates satisfying <`filterExpr`> (2-D only).

Table 1.26: `NOTICE` Command Arguments

The typical <`filterExpr`> for 1-D data is the *range* filter:

This Filter Expression:	**Means:**
`#:#`	Include all data from the first value to the second value, inclusive.
`#:`	Include all data above the given value, inclusive.
`:#`	Include all data below the given value, inclusive.
`#`	Include the data point corresponding to the given value.

Table 1.27: Specification of Range Filters for 1-D Data

For 2-D data, <filterExpr> is a quote-delimited region specification. See “ahelp dmregions” for more information on allowed region specifications.

For more information on filter expressions, see the *Sherpa Filtering* Chapter for the full <`filterExpr`> definition, as well as information regarding other filtering methods.

The difference between the arguments `FILTER` and {`BINS` | `CHANNELS`} may be illustrated with the following simplistic ASCII dataset:

```
x       y
-       -
22.6    10
22.7    9
22.8    12
22.9    7
```

The command `NOTICE FILTER 22.65:22.85` will cause the middle two bins to be included in the filter. This is equivalent to `NOTICE BINS 2:3`, *i.e.*, notice the 2nd and 3rd bins. For PHA datasets, `NOTICE FILTER` and `NOTICE BINS` will lead to the same result, since the x-values of the PHA channels are sequential integers: 1, 2, 3...

To use the commands `NOTICE ENERGY` or `NOTICE WAVE`, a PHA dataset must have been read in, and an `INSTRUMENT` model stack defined. Note that regardless of the current `ANALYSIS` setting, one may filter using either `NOTICE ENERGY` or `NOTICE WAVE`. See the examples below.

To specify regions to notice within an image:

- Display the image and mark the filter region(s) (see the `IMAGE` command). *NOTE:* only **`include`** regions are allowed for filtering (as opposed to **`exclude`** regions). Instead of using `NOTICE IMAGE` with an **`exclude`** region, use `IGNORE IMAGE` with an **`include`** region instead.

  Verify results using `IMAGE FILTER`.

- Issue the command `IGNORE` [<**`dataset range`**> | `ALLSETS`] `ALL`. *Note that if this is the first type the* ***`NOTICE`*** *command below is to be issued for a particular dataset, the user need not issue the command* ***`IGNORE ALL`****.*

- Issue the command `NOTICE` [<**`dataset range`**> | `ALLSETS`] `IMAGE`. This will tell *Sherpa* to include data within the marked region(s).

To specify regions to notice within a plot:

- Plot the data using the `LPLOT` command).

- Issue the command `IGNORE` [<**`dataset range`**> | `ALLSETS`] `ALL`. *Note that if this is the first type the* ***`NOTICE`*** *command below is to be issued for a particular dataset, the user need not issue the command* ***`IGNORE ALL`****.*

- Issue the command `COMMAND` [<**`dataset range`**> | `ALLSETS`] `PLOT`.

- Place the cursor on the plot, at the desired minimum x-axis value for the filter, and click once with the left mouse button (or hit the `e` key).

- Place the cursor at the desired maximum x-axis value, and click again with the left mouse button (or hit the `e` key). *Sherpa* will then include data within the marked region(s).

Ranges/regions to be notices may be alternatively set using the *Sherpa*/S-Lang module functions `set_notice` and `set_notice2d`, *et al.*

Examples:

1. Include all data values:

   ```
   sherpa> DATA example.dat
   sherpa> NOTICE ALL
   ```

   The last command tells Sherpa to use all data in dataset 1.

2. Include a particular data range from the source, background, or both:

   ```
   sherpa> DATA data/data.dat
   sherpa> BACK data/data_bckg.dat
   sherpa> IGNORE SOURCE ALL
   sherpa> NOTICE SOURCE FILTER 5:10
   sherpa> LPLOT 2 DATA BACK
   ```

   After the input of source and background data, the command `NOTICE SOURCE FILTER 5:10` tells *Sherpa* to include those bins in the *source* dataset whose x-axis values are equal to 5 through 10, inclusive. Note that the preceding `IGNORE SOURCE ALL` command is not strictly necessary, but is included here for completeness. See the *Sherpa Filtering* Chapter for further examples of filter expressions and usage of the `IGNORE` command. With the final command, a plot of both the source and background data illustrates that the filter was applied to the source data only; the background data remains unfiltered.

   ```
   sherpa> NOTICE ALL
   sherpa> IGNORE BACK ALL
   sherpa> NOTICE BACK FILTER 5:10
   sherpa> LPLOT 2 DATA BACK
   sherpa> LPLOT BFILTER
   ```

   The `NOTICE ALL` command eliminates any filters applied to the source and background datasets. The command `NOTICE BACK FILTER 5:10` tells *Sherpa* to include those bins in the *background* dataset whose x-axis values are equal to 5 through 10, inclusive. Note that the preceding `IGNORE BACK ALL` command is not strictly necessary, but is included here for completeness. With the first `LPLOT` command, a plot of both the source and background data illustrates that the filter was applied to the background data only; the source data remains unfiltered. The final command plots the filter status of each background data point.

   ```
   sherpa> IGNORE ALL
   sherpa> NOTICE FILTER 5:10
   sherpa> LPLOT 2 DATA BACK
   sherpa> LPLOT FILTER
   ```

   The `NOTICE FILTER 5:10` command tells *Sherpa* to exclude those bins in both the source and background datasets whose x-axis values are equal to 5 through 10, inclusive. Note that the preceding `IGNORE ALL` command is not strictly necessary, but is included here for completeness. With the first `LPLOT` command, a plot of both the source and background data illustrates that the filter was applied to both datasets. The final command plots the filter status of each source data point.

3. Include a 2-D region, specified from an image display:

```
sherpa> READ DATA example_img.fits
sherpa> IMAGE DATA
sherpa> IGNORE ALL
<mark include region(s)>
sherpa> NOTICE IMAGE
sherpa> IMAGE FILTER
```

   In this example, 2-D image data is first displayed. Then the data filter is set to exclude the entire image, with the command `NOTICE ALL`. (This command is not needed in this particular example but is shown for completeness.) Next, the user chooses regions, by placing `include` markers on the displayed image. The command `NOTICE IMAGE` sets the data filter to exclude data within the marked regions. The command `IMAGE FILTER` displays the resulting filter.

4. Include a 1-D data range, specified from a plot display:

```
sherpa> READ DATA example.dat
sherpa> LPLOT DATA
sherpa> IGNORE ALL
sherpa> NOTICE PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER
sherpa> LPLOT DATA
```

   In this example, 1-D data is first displayed. Then the data filter is set to exclude the entire dataset, with the command `IGNORE ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `NOTICE PLOT` readies the cursor for selecting the desired filter region, and the user should then left-mouse-click first at the desired minimum x-axis location and then again at the maximum x-axis location. After the second left-mouse-click, the *Sherpa* command prompt is returned. The command `LPLOT FILTER` will then plot the data region that you have marked for inclusion. Finally, the dataset is plotted again, with the command `LPLOT DATA`, showing that the selected regions have been properly included.

5. Include a 1-D data range, specified from a plot display, for dataset number 2:

```
sherpa> READ DATA 1 example1.dat
sherpa> READ DATA 2 example2.dat
sherpa> LPLOT DATA 2
sherpa> IGNORE 2 ALL
sherpa> NOTICE 2 PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER 2
sherpa> LPLOT DATA 2
sherpa>
sherpa> IGNORE ALL
sherpa> NOTICE PLOT
<left-mouse-click once at the desired minimum x-axis value>
<and then left-mouse-click again at the desired maximum x-axis value>
sherpa> LPLOT FILTER
sherpa> LPLOT DATA
```

In this example, two 1-D datasets are input, and dataset number 2 is first displayed. Then, the data filter is set to exclude all of dataset number 2, with the command `IGNORE 2 ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `NOTICE 2 PLOT` readies the cursor for selecting the desired filter region for dataset number 2, and the user should then left-mouse-click first at the desired minimum x-axis location and then again at the maximum x-axis location. After the second left-mouse-click, the *Sherpa* command prompt is returned. The command `LPLOT FILTER 2` will then plot the data region that you have marked for inclusion from dataset number 2. Finally, dataset number 2 is plotted again, with the command `LPLOT DATA 2`, showing that the selected regions have been properly included.

Next, the user sets the data filter to exclude all of dataset number 1, with the command `IGNORE ALL`. (This command is not needed in this particular example but is shown for completeness.) The command `NOTICE PLOT` readies the cursor for selecting the desired filter region for dataset number 1, and the user should then left-mouse-click first at the desired minimum x-axis location and then at the maximum x-axis location (from the plot that's currently displayed of dataset number 2). The command `LPLOT FILTER` will then plot the data region that you have marked for inclusion from dataset number 1. Finally, dataset number 1 is plotted again, with the command `LPLOT DATA`, showing that the selected regions have been properly included. Note that dataset number 1 was interactively filtered, *from a display of dataset number 2.*

6. Include particular data values and ranges:

```
sherpa> DATA 3 example.dat
sherpa> IGNORE 3 ALL
sherpa> NOTICE 3 FILTER 4, 8:, 1:3
```

The first command, `IGNORE 3 ALL`, sets all of dataset number 3 to be excluded. (This command is not needed in this particular example but is shown for completeness.) The second command uses a filter expression to include those x-axis data values that are equal to 4, greater-than or equal to 8, or equal to 1 thru 3, inclusive. See the *Sherpa Filtering* Chapter for further examples of filter expressions and usage of the `NOTICE` command.

7. Filter PHA data by energy, or wavelength:

```
sherpa> ERASE ALL
sherpa> DATA example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
<directory_path>/example.rmf
ARF is being input from:
<directory_path>/example.arf
Background data are being input from:
<directory_path>/example_bkg.pha
sherpa> ANALYSIS ENERGY
sherpa> IGNORE ALL
sherpa> NOTICE ENERGY 5:10
sherpa> LPLOT DATA
sherpa> IGNORE ALL
sherpa> NOTICE WAVE 0:20
sherpa> ANALYSIS WAVE
sherpa> LPLOT DATA
```

In this example, the dataset is filtered using NOTICE WAVE even though the ANALYSIS setting is ENERGY. This is possible because header information in the data file allowed for an instrument model to be automatically defined when the data were initially read.

8. Include a 2-D region specified from the command line, in physical coordinates:

```
sherpa> DATA data/example_img2.fits
sherpa> NOTICE PHYSICAL "CIRCLE(4010,3928,100)"
sherpa> IMAGE FILTER
```

To filter in physical coordinates requires no action on the part of the user beyond specifying the PHYSICAL modifier; *Sherpa* automatically performs the image-to-physical coordinate conversion (if it can). Note that the quote marks are required in CIAO 3.0 when specifying 2-D filter regions. (They are not required when specifying 1-D regions, as seen in the other examples above.) The command IMAGE FILTER displays the resulting filter.

1.56 OPEN

Opens the image display window.

```
sherpa> OPEN IMAGE
```

where IMAGE is the imaging window (see the IMAGE command).

Note: If there is no open imaging window, giving any IMAGE command will lead to the creation of one. Similarly, if there is no open plotting window, giving a LPLOT, CPLOT, or SPLOT command will lead to the creation of one.

See the CLOSE command for information on closing an image display window. Note also that image display windows, as well as plotting display windows, may be closed simply by closing the window with a mouse click.

See the Sherpa Display Chapter for further information regarding data display capabilities within *Sherpa*.

Examples:

1. Open and close the imaging window:

```
sherpa> OPEN IMAGE
sherpa> CLOSE IMAGE
```

1.57 OPLOT

Causes multiple data curves to be displayed in the same drawing area, via *ChIPS*.

```
sherpa> OPLOT <arg_1> [# [ID]] [<arg_2> [# [ID]] ...]
```

`#` specifies the number of the dataset (default is 1). The `ID` modifier is used to display background datasets, if and only if the *Sherpa* state object variable `multiback` is set to 1 (*i.e.* if more than one background dataset is to be associated with a single source dataset). The `ID` modifier may be any string that is not a parsable command (*e.g.* "A", "bob", "foo").

The allowed arguments <**arg_n**> are listed in the documentation for `LPLOT`. The exceptions to that information are the `FIT` plots: if used in the `OPLOT` command, only the data is displayed, not the data and the model.

The appearance of plots generated with this command can be changed by modifying the fields of certain state objects. See the ahelp for *Sherpa* or for sherpa.plot for more information.

When using *ChIPS* commands to modify plot characteristics from within *Sherpa*, the `REDRAW` command must be issued in order to view the changes.

Examples:

1. Overplot two sets of 1-D data:

   ```
   sherpa> DATA data/example1.dat
   sherpa> DATA 2 data/example2.dat
   sherpa> OPLOT DATA 1 DATA 2
   ```

2. In this example, a fit is read in from a saved *Sherpa* session. Then the data, individual model components of the multi-component model, and the sum of the individual model components are plotted:

   ```
   sherpa> use fit.shp
   sherpa> oplot ufit source galabs powhard
   sherpa> c 2 red
   sherpa> c 3 green
   sherpa> c 4 blue
   sherpa> log
   sherpa> redraw
   ```

   The model components "galabs" and "powhard" are defined in fit.shp. After plotting, the curves are given different colors to differentiate them, and the plot is changed to log scale.

1.58 PARAMPROMPT

Turns on/off prompting for model parameter values.

```
sherpa> PARAMPROMPT {ON | OFF}
```

By default, prompting for model parameter values is `ON`. However, when an *Sherpa* script file name is provided on the UNIX command line (*e.g.* `unix> sherpa example.script`) or is provided as the argument of a `USE` command, Section 1.89, prompting for model parameter values is automatically turned off.

For more information about parameter prompting, see the `CREATE` command, Section 1.23.

Examples:

1. Turn off parameter prompting; set parameter values using the model language syntax:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE 2 = POW[modele]
sherpa> modele.ampl = 2.3
```

2. Call and execute *Sherpa* script; set parameter values using the model language syntax:

```
sherpa> $more example.script
# This is my sherpa script
READ DATA example.dat
POLY[modela]
SOURCE = modela
FIT

sherpa> USE example.script
 powll: v1.2
 powll:   initial function value =     3.25453E+02
 powll:     converged to minimum =     9.80351E+01 at iteration =      2
 powll:   final function value    =     9.80351E+01
        modela.c0  10.4657
sherpa>
```

This example shows that when a *Sherpa* script is called and executed, prompting for model parameter values is automatically turned off.

1.59 PLOTX

[REMOVED AS OF CIAO 3.0.2] Sets the unit type for the x-axis of a plot.

This command was removed from CIAO at version 3.0.2.

The `PLOTX` command has been disabled in order to lessen confusion in visualization. Plots are done using the `ANALYSIS` setting, if applicable, and in channels otherwise; see "ahelp analysis" for more information.

NOTE: A bug in CIAO 3.1 does not allow for plotting the PHA data in the channel space when the instrument model is specified.

1.60 PLOTY

Sets the unit type for the y-axis of a plot.

```
sherpa> PLOTY [#] {COUNTS | RATE || ONE | ENERGY | WAVE | ENERGY2 | WAVE2 }
```

where `#` may specify the number of the dataset; the default dataset is 1.

The command `PLOTY` actually controls two aspects of plot appearance: whether rates or raw data are shown; and whether the amplitudes in each bin are to be multiplied by the energy/wavelength of the bin.

1.60.1 RATE vs. COUNTS

If the user specifies `COUNTS`, the raw input data will be displayed (even if these data are not actually counts, but something else, such as fluxes). What is displayed if the argument `RATE` is given depends on whether:

- the exposure time is provided;
- an ancillary response file (ARF) is provided;
- the energy or wavelength grid is provided; and/or
- the instrument model is defined (see the `INSTRUMENT` command, Section 1.46).

Argument:	Description:
`ONE`	No multiplication by energy or wavelength.
`ENERGY`	Multiply the amplitudes in each bin by the bin energies.
`WAVE`	Multiply the amplitudes in each bin by the bin wavelengths.
`ENERGY2`	Multiply the amplitudes in each bin by the bin energies squared.
`WAVE2`	Multiply the amplitudes in each bin by the bin wavelengths squared.

Table 1.28: The user can further customize plots using one of the following `PLOTY` settings:

The plot labeling function assumes exposure times in seconds, ARFs with effective area information in cm^2, and energies/wavelengths in keV/Å. If necessary, the user can change the label with the *ChIPS* command `YLABEL`.

Note that in CIAO 3.0, the user cannot have separate `PLOTY` settings for source and background datasets with the same dataset `#`, *i.e.*, changing `PLOTY` changes the plot appearance for *both* the source *and* the background datasets.

Also note that `PLOTY` controls not just the appearance of 1D plots, but also the output generated by the `WRITE` command. This may be changed in a future version of *Sherpa*.

Examples:

1. Reset the y-axis units to counts, after they have automatically been set to rate with the definition of an instrument model:

```
sherpa> READ DATA data/example.pha
sherpa> RSP[instrumentA](example.rmf, example.arf)
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> INSTRUMENT = instrumentA
sherpa> LPLOT DATA
<Rate data are displayed.>
sherpa> PLOTY COUNTS
sherpa> LPLOT DATA
<Counts data are displayed.>
```

1.61 PROJECTION

Estimates confidence intervals for selected thawed parameters.

```
sherpa> PROJECTION [<dataset range> | ALLSETS] [ <arg_1> , ... ]
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to estimate limits using data from all appropriate datasets.

The command-line arguments <arg_n> may be:

Argument	**Description**
<`sherpa_modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `GAUSS.pos`).
<`modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.29: `PROJECTION` Command Arguments

The user may configure `PROJECTION` via the *Sherpa* state object structure `proj`. The current values of the fields of this structure may be displayed using the command `print(sherpa.proj)`, or using the more verbose *Sherpa*/S-Lang module function `list_proj()`.

The structure fields are:

Field	**Description**

`fast`	If 1, use a fast optimization algorithm (`LEVENBERG-MARQUARDT` or `SIMPLEX`) regardless of the current `METHOD`. If 0, use the current `METHOD`.
`sigma`	Specifies the number of σ (*i.e.*, the change in statistic).

Table 1.30: `proj` Structure Fields

Field values may be set using directly, *e.g.*,

```
sherpa> sherpa.proj.sigma = 2.6
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_proj()`.

Confidence interval bounds are determined for each selected parameter in turn. A given parameter's value is varied along a grid of values while the values of all the other nominally thawed parameters are allowed to float to new best-fit values (compare to `UNCERTAINTY` (Section 1.85), where the values of all the other nominally thawed parameters remain fixed to their best-fit values). This method of estimating confidence interval bounds gives truly accurate results only in special cases (see caveats below).

Because `PROJECTION` estimates confidence intervals for each parameter independently, the relationship between `sigma` and the change in statistic value ΔS can be particularly simple: $\sigma = \sqrt{\Delta S}$ for statistics sampled from the χ^2 distribution and for the Cash statistic (Section 6.3), and is approximately equal to $\sqrt{2\Delta S}$ for fits based on the general log-likelihood ($\log \mathcal{L}$).

Confidence	σ	$\Delta\chi^2$	$\Delta \log \mathcal{L}$
68.3%	1.0	1.00	0.50
90.0%	1.6	2.71	1.36
95.5%	2.0	4.00	2.00
99.0%	2.6	6.63	3.32
99.7%	3.0	9.00	4.50

Table 1.31: Confidence Intervals for the `projection` command

1.61.1 Caveats

An estimated confidence interval is accurate if and only if:

1. the χ^2 or $\log\mathcal{L}$ surface in parameter space is approximately shaped like a multi-dimensional paraboloid, and

2. the best-fit point is sufficiently far from parameter space boundaries.

One may determine if these conditions hold, for example, by plotting the fit statistic as a function of each parameter's values (the curve should approximate a parabola) and by examining contour plots of the fit statistics made by varying the values of two parameters at a time (the contours should be elliptical, and parameter space boundaries should be no closer than approximately 3σ from the best-fit point). The `INTERVAL-PROJECTION` (Section 1.48) and `REGION-PROJECTION` (Section 1.66) commands can be used for these checks.

If either of these conditions does not hold, **then the output from `PROJECTION` may be meaningless except to give an idea of the scale of the confidence intervals**. To accurately determine the confidence intervals, one would have to reparameterize the model, or use Monte Carlo simulations or Bayesian methods.

The user may retrieve the upper- and lower-bound estimates using the *Sherpa*/S-Lang module function `get_proj`.

Examples:

1. List the current and default values of the `proj` structure, and restore the default values:

```
sherpa> sherpa.proj.sigma = 5
sherpa> list_proj()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
sigma             5               1                    Number of sigma
sherpa> restore_proj()
sherpa> list_proj()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
sigma             1               1                    Number of sigma
```

2. Search parameter space to find a range of parameter values, within a confidence in terval of 68.3%, for all thawed parameters:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = poly[p]
sherpa> THAW p.c1 p.c2 p.c3
sherpa> METHOD SIMPLEX
sherpa> FIT
...
sherpa> PROJECTION
Projection: optimization reset to LM.
Projection complete for parameter: p.c0
Projection complete for parameter: p.c1
Projection complete for parameter: p.c2
Projection complete for parameter: p.c3

Computed for projection.sigma = 1
```

```
-------------------------------------------------------
Parameter Name      Best-Fit Lower Bound     Upper Bound
-------------------------------------------------------
     p.c0         -0.302766  -8.34233        +7.87706
     p.c1          0.598026  -8.7713         +9.42289
     p.c2          0.792121  -2.72725        +2.53915
     p.c3          0.018429  -0.205409       +0.2213
```

1.61.2 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `PROJECTION` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `PROJECTION` will not be valid unless the parameters are at their best-fit values.

1.62 PROMPT

Changes the *Sherpa* prompt.

```
sherpa> PROMPT "<arg>"
```

where <`arg`> is the string that will replace the current Sherpa prompt.

Examples:

1. Change the *Sherpa* prompt:

```
sherpa> PROMPT "hello> "
hello>
```

2. Return the *Sherpa* prompt to the default:

```
hello> PROMPT "sherpa> "
sherpa>
```

1.63 QUIT — BYE — EXIT

Terminates the *Sherpa* program.

See the `BYE` command (Section 1.17).

1.64 READ

Inputs the contents of one or more files.

```
sherpa> READ <arg> [# [ID]] <filespec> [,[# [ID]] <filespec>,...]
```

where `#` specifies the number of the dataset to be associated with the data file (default dataset number is 1). The `ID` modifier is used only when background data are input, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

<**`arg`**> may be:

Argument	**To input a file containing:**
{`DATA` | `BACK`}	Source|background data values
{`ERRORS` | `BERRORS`}	Estimated total errors for the source|background data
{`SYSERRORS` | `BSYSERRORS`}	Systematic errors for the source|background data
{`WEIGHT` | `BWEIGHT`}	Statistic weight values assigned to each source|background data point
{`FILTER` | `BFILTER`}	Mask values (0|1) for each source|background data point
{`GROUP` | `BGROUP`}	Grouping values (1|-1) associated with each source|background data point
{`QUALITY` | `BQUALITY`}	Quality values (0|2|5) associated with each source|background data point
`MDL`	A model descriptor list file.

Table 1.32: `READ` Command Arguments

Sherpa currently supports the following file types:

<**`filetype`**>	**Containing:**
`ASCII`	ASCII data
`FITS`	FITS image data
`FITSIMAGE`	FITS image data
`FITSBIN`	FITS binary table data
`IMH`	*IRAF* image data
`PHA`	Pulse-height amplitude data

Table 1.33: `READ` File Type Arguments

For each of these file types, we discuss the allowed <**`filespec`**> arguments in turn.

1.64.1 ASCII File Type:

<**`filename`**> [{`ASCII` | `HISTOGRAM`}] [<**`colnumbers`**>]

where

`<filename>`	The name of the data file (this may also include a path).
`<colnumbers>`	A list of column numbers.

The modifier `ASCII` indicates that the input data is to be treated as unbinned (*i.e.*, models are to be evaluated at single points), while `HISTOGRAM` leads *Sherpa* to create a binned dataset from the ASCII file. This is done by treating the first data point as the lower bin boundary for the first channel, the second data point as the upper bin boundary for the first channel and the lower bin boundary for the second channel, *etc.* The last data point (which has no defined upper boundary in this scheme) is dropped. This command is particularly useful when using additive *XSPEC* models, since these models are always integrated over a bin and so require binned data.

Note that when reading an ASCII file containing more than two columns, only the data in the first two columns are input. To read other columns, or more than two columns, specify the column numbers with `<colnumbers>`. See the examples below.

Also note that the last column input is considered the dependent coordinate.

FITS, FITSIMAGE, IMH, and QP File Types:

1.64.2 FITS, FITSIMAGE, IMH, and QP File Types:

`["]<filename>[<virtual_file_syntax>"] [{FITS | FITSIMAGE | IMH | QP}]`

where

`<filename>`	The name of the data file (this may also include a path).
`<virtual_file_syntax>`	A filtering and/or binning command argument. See the "Using Data Model Filters" section for further information.

If the command argument `{FITS | FITSIMAGE}` is not included when reading a FITS file, *Sherpa* will attempt to determine the FITS file type (*e.g.*, 2-D FITS image *vs.* 1-D FITS binary table) from the FITS header keywords.

Note that whenever `<virtual_file_syntax>` is specified, `<filename><virtual_file_syntax>` usually must be surrounded by quotes, `" "`.

FITSBIN (FITS Binary Table) File Type:

1.64.3 FITSBIN (FITS Binary Table) File Type:

`"<filename><virtual_file_syntax>" [FITSBIN]`

where

<**filename**>	The name of the data file (this may also include a path).
<**virtual_file_syntax**>	A filtering and/or binning command argument. See the "Using Data Model Filters" section for further information.

Note that <**virtual_file_syntax**> should be included, in order to specify the desired columns. Otherwise, *Sherpa* will try to input data from *all* columns, which will lead to an error message if there are more than two columns.

Also note that if the command argument FITSBIN is not included, *Sherpa* will attempt to determine the FITS file type (*e.g.*, 2-D FITS image *vs.* 1-D FITS binary table) from the FITS header keywords.

Last, note that whenever <**virtual_file_syntax**> is specified, <**filename**><**virtual_file_syntax**> usually must be surrounded by quotes, " ".

PHA File Types (Types I and II):

1.64.4 PHA File Types (Types I and II):

<**filename**> [PHA]

where

<**filename**>	The name of the data file (this may also include a path).

Note that if a FITS binary table is input with no <**filetype**> modifier [FITSBIN or PHA], the file type will be automatically inferred; if the file contains an extension named SPECTRUM, it is assumed to be a PHA file.

If the input PHA file contains a GROUPING column, the data are automatically grouped. Also, *Sherpa* retains the contents of the QUALITY columns, allowing the user to, *e.g.*, filter out bad channels by issuing the command IGNORE BAD.

If the input PHA file contains a STAT_ERR column, its contents are ignored (and a message printed to the screen). If you wish to use these statistical error estimates (as opposed to letting *Sherpa* estimate the errors given a chosen STATISTIC, you should read them in as follows (substituting BERRORS if appropriate):

```
sherpa> READ ERRORS "<filename>[cols CHANNEL,STAT_ERR] FITSBIN
```

On the other hand, if the input PHA file contains a SYS_ERR column, its contents are used; see SYSERRORS for more information.

If the STAT_ERR is read in before a fit, and SYS_ERR was also read in, then the error in a bin is $\sqrt{\mathrm{STAT\_ERR}^2 + \mathrm{SYS\_ERR}^2}$.

If STAT_ERR is not read in, and SYS_ERR is read in, then the error in a bin is $\sqrt{[\text{computed Poisson error}]^2 + \texttt{SYS\_ERR}^2}$.

If the header of the input PHA file includes keywords that references background data files, or source and/or background response files, then these files and automatically read in and, if appropriate, RSP instrument models are automatically defined (See INSTRUMENT command, Section 1.46 for more information). When these files are read in, the following kinds of messages are issued:

```
RMF is being input from:
  <directory_path>/example.rmf
ARF is being input from:
  <directory_path>/example.arf
Background data are being input from:
  <directory_path>/example_bkg.pha
Background RMF is being input from:
  <directory_path>/example_bkg.rmf
Background ARF is being input from:
  <directory_path>/example_bkg.arf
```

Note the following, for all file types:

- Beginning with *Sherpa* version 3.0.2, it is no longer required that multi-dimensional data be defined on a uniform rectangular grid (*e.g.*, such as is the case in a FITS image). The user can, for instance, input amplitudes at arbitrary coordinates and fit models to these amplitudes. (However, the analysis of arbitrarily placed *binned* [*histogramed*] data is not supported.)
- Background datasets and errors, source dataset errors, and filters, read in *prior* to the entry of source data are erased when the source data are read in.
- The data input from ASCII and FITS files can be of arbitrary dimensionality (*e.g.*, the user can read four columns from an ASCII file to create a 3-D dataset, or can read a 1-D FITS image, *etc.*).
- One uses READ FILTER to read in filter values for each bin or pixel, and *not* to read in, for instance, region descriptors. These filter values should be either 1 for inclusion, or 0 for exclusion, of the data point. See the example below, as well as the descriptions of the related commands IGNORE and NOTICE.
- To use input data as a model, do not use the READ command; rather, use the GRIDMODEL model.
- Inputting files of type QP and IMH is not supported for non-Solaris platforms.

1.64.5 Using Data Model Filters

This command is able to take any Data Model virtual file specification (see "ahelp dmsyntax"). If you can do

```
unix% dmcopy "infile.fits[spec 1][spec 2]" outfile.fits
```

you can also do

```
sherpa> read "infile.fits[spec 1][spec 2]"
```

This is especially useful when working with very large files. For example:

```
sherpa> read "evt.fits[bin sky=4][opt mem=100]"
```

bins the event file by a factor of four and allocates additional memory. A similar command (omitting the binning factor) can be used to read in an image.

Examples:

1. Input an ASCII data file having a .dat extension name:

   ```
   sherpa> READ DATA 1 example.dat ASCII 1 2
   ```

   Reads the first two columns of the ASCII data file `example.dat`, as dataset number 1. The following commands are each equivalent to the above command:

   ```
   sherpa> READ DATA 1 example.dat ASCII
   sherpa> READ DATA 1 example.dat
   sherpa> READ DATA example.dat
   sherpa> DATA example.dat
   ```

   Note that, if not specified, only the first two columns are read. Also, the dataset number is assumed to be 1 if it is not specified.

2. Input ASCII data and error files not having a .dat extension name:

   ```
   sherpa> READ DATA 1 example.qdp ASCII 1 2
   sherpa> READ ERRORS 1 example.qdp ASCII 1 3
   ```

   The first command reads columns 1 and 2 of the ASCII data file `example.qdp`, as dataset number 1. Then, columns 1 and 3 of the same ASCII data file are read, as the measurement errors of this dataset. Note that using the `ASCII` argument is no longer necessary for input of files not having a `.dat` extension. Thus, the following `READ DATA` commands are each equivalent to the above `READ` command:

   ```
   sherpa> READ DATA 1 example.qdp ASCII
   sherpa> READ DATA 1 example.qdp
   sherpa> READ DATA example.qdp
   sherpa> DATA example.qdp
   ```

3. Input various data columns from ASCII data and error files:

   ```
   sherpa> READ DATA 1 example.dat 3 8
   sherpa> READ ERRORS 1 example.dat 3 5
   sherpa> READ DATA 2 example.dat
   sherpa> READ ERRORS 2 example.dat 1 4
   ```

The first command reads columns 3 and 8 of the ASCII data file `example.dat`, as dataset number 1. Next, the measurement errors for dataset 1, from columns 3 and 5 of `example.dat` are read. Then, the first and second columns of the ASCII data file `example.dat`, as dataset number 2, are read. The last command reads the measurement errors for dataset 2, from columns 1 and 4 of example.dat.

4. Overwrite dataset number 1:

```
sherpa> READ DATA 1 example1.dat 3 8
sherpa> READ DATA example2.dat
```

Note that the command `READ DATA example2.dat` overwrites the data that had been input from `example1.dat`.

5. Input multiple ASCII datasets, using a single command:

```
sherpa> READ DATA 1 example1.dat, 2 example2.dat, 3 example3.dat 2 3
```

This example illustrates the input of multiple data files simultaneously. The command reads `example1.dat` as dataset number 1, and example2.dat as dataset number 2. Columns 2 and 3 of `example3.dat` are read as dataset number 3. The following command is equivalent:

```
sherpa> READ DATA example1.dat, example2.dat, example3.dat 2 3
```

6. Input ASCII data, and weight assignments from a file:

```
sherpa> READ DATA example1.dat 1 2
sherpa> READ WEIGHT example1.dat 1 3
sherpa> SHOW WEIGHTS
```

The first command reads columns 1 and 2 of the ASCII data file `example1.dat`, as dataset number 1. Column 3 of `example1.dat` contains a weight assignment for each of the data points. These weight assignments are input with the second command. Current weight assignments for each data point can be reported with the command `SHOW WEIGHTS`.

7. Input ASCII data, and filter assignments from a file:

```
sherpa> READ DATA example1.dat 1 2
sherpa> READ FILTER example1.dat 1 3
sherpa> SHOW FILTER
```

The first command reads columns 1 and 2 of the ASCII data file `example1.dat`, as dataset number 1. Column 3 of `example1.dat` contains a filter assignment for each of the data points (1 for the data point to be included; 0 for the data point to be excluded). Current filter assignments for each data point can be reported with the command `SHOW FILTER`.

8. Compare input ASCII data to input HISTOGRAM data:

```
sherpa> READ DATA 1 data/example.dat
sherpa> SHOW DATA 1
Y Column: Counts
  Dimensions: 1
```

```
  Total Size: 4 bins (or pixels)
  Axis: 0; Name: Bin
  Length: 4 bins (or pixels)
File Name: data/example.dat
SubSection (if any):
File Type: ASCII
[1] = 1
[2] = 5
[3] = 8
[4] = 17

sherpa> READ DATA 2 data/example.dat HISTOGRAM
sherpa> SHOW DATA 2
Y Column: Counts
  Dimensions: 1
  Total Size: 3 bins (or pixels)
  Axis: 0; Name: Bin
  Length: 3 bins (or pixels)
File Name: data/example.dat
SubSection (if any):
File Type:
[1.500000] = 1
[2.500000] = 5
[3.500000] = 8
```

9. Utilize the HISTOGRAM argument to input binned data:

```
sherpa> READ DATA 1 data/spectrum_notintegrated.dat ASCII
```

The above command inputs data from an ASCII file that has two columns: energy (in keV), and flux (in photons/cm^2/sec/keV). Note that this dataset cannot be used with additive *XSPEC* models, since they require binned data.

```
sherpa> READ DATA 2 data/spectrum_integrated.dat HISTOGRAM
```

The above command inputs and bins data from an ASCII file that has two columns: energy (in keV), and flux (in photons/cm^2/sec). (*i.e.*, where the second column contains data of units photons/cm^2/sec/keV multiplied by the bin width in keV). Note that this dataset can be used with *XSPEC* models, since the input data are binned.

10. Input a 2-D FITS image data file:

```
sherpa> READ DATA 1 example_img.fits FITS
```

This command reads the 2-D FITS image `example_img.fits` as dataset number 1. The following command is equivalent:

```
sherpa> READ DATA 1 example_img.fits FITSIMAGE
```

The following commands are also equivalent to the above, if the `example_img.fits` file contains the proper header kewords identifying the file as a 2-D FITS image:

```
sherpa> READ DATA example_img.fits
sherpa> DATA example_img.fits
```

11. Input 2-D FITS image data and background files:

```
sherpa> READ DATA 3 example_img.fits FITSIMAGE
sherpa> READ BACK 3 example_img_bkg.fits FITSIMAGE
```

First, the FITS image `example_img.fits`, as dataset number 3, is read. Then, the background FITS image for this dataset is read. Note that the command `SUBTRACT` must be issued in order to actually have the background subtracted from the data.

12. Input a portion of a 2-D FITS image data file:

```
sherpa> READ DATA "example_img.fits[#1=100:200, #2=100:400]" FITS
```

This command reads the specified portion of the 2-D FITS image `example_img.fits`. Note that, by default, the data is taken from the first FITS block for which NAXIS is nonzero. The following command is equivalent:

```
sherpa> READ DATA "example_img.fits[100:200,100:400]" FITS
```

13. Input various data columns from a FITS binary data file:

```
sherpa> READ DATA 1 "example_bin.fits[2][columns #1, #2]" FITSBIN
```

This command reads the first two columns from the second extension of the FITS binary table file `example_bin.fits`, as dataset number 1. Note that column numbers or names must always be specified when reading FITS binary table files. The following commands are each equivalent to the above command:

```
sherpa> READ DATA 1 "example_bin.fits[2][cols #1, #2]" FITSBIN
sherpa> DATA "example_bin.fits[2][cols #1, #2]" FITSBIN
```

14. Input various data columns from FITS binary data files:

```
sherpa> READ DATA 2 "example_bin.fits[2][cols TIME, EXPNO]" FITSBIN
sherpa> READ DATA 3 "example_bin.fits[EVENTS][cols time, expno]" FITSBIN
```

The first command reads columns time and expno, from the second extension of the FITS binary table file `example_bin.fits`, as dataset number 2. The second command reads columns time and expno, from the EVENTS extension of the FITS binary table file `example_bin.fits`, as dataset number 3. Note that columns may be specified by case insensitive name. Also, the FITS extension can be specified by either the number or the name of the extension (in this example, the name of the second extension is EVENTS).

15. Input a 2-D image by binning columns from a FITS binary data file:

```
sherpa> DATA "example_bin.fits[bin chipx, chipy]" FITSIMAGE
Warning: Could not retrieve WCS coord descriptor
```

This command inputs into *Sherpa* the FITS binary table `example_bin.fits`, but bins the table to create an image using the `chipx` and `chipy` columns. Note that the `FITSIMAGE` argument is required since it is ultimately a FITS image that is being input to *Sherpa*.

16. Input a 2-D image by binning and filtering columns from a FITS binary data file:

```
sherpa> DATA "example_bin.fits[bin chipx=200:400:4, chipy=300:400:4]" FITSIMAGE
```

This command creates and inputs into *Sherpa*, an image using the `chipx` and `chipy` columns of the binary FITS table file `example_bin.fits`. In this example, ranges for the axes, and bin sizes, are given. Note that the `FITSIMAGE` argument is required.

17. Input a 2-D image data file:

```
sherpa> READ DATA 1 example.imh IMH
```

This command reads the image `example.imh`, as dataset number 1. The following commands are each equivalent:

```
sherpa> READ DATA example.imh IMH
sherpa> READ DATA example.imh
sherpa> DATA example.imh
```

18. Input a portion of a 2-D image data file:

```
sherpa> READ DATA 2 "example.imh[#1=100:200, #2=100:400]"
```

This command reads a portion of the image `example.imh`, from Axis 0 coordinates 100 to 200 and from Axis 1 coordinates 100 to 400. The data are read as dataset number 2. The following command is equivalent:

```
sherpa> READ DATA 2 "example.imh[100:200, 100:400]"
```

19. Input PHA data and background files:

```
sherpa> READ DATA 4 example.pha PHA
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example2.rmf
ARF is being input from:
  <directory_path>/example2.arf
Background data are being input from:
  <directory_path>/example2_bkg.pha
sherpa> READ BACK 4 example_bkg.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
```

First, the PHA data file `example.pha` is read as dataset number 4. Note that the systematic errors contained in the PHA data file are input. These input systematic errors are added in quadrature with the statistical errors (which are automatically computed using the currently defined `STATISTIC`). Note also that since the header of the PHA data file contains the proper keywords, instrument and background data files are automatically loaded. The `READ BACK 4 example_bkg.pha` command inputs the background PHA file is read for dataset number 4.

20. Input multiple PHA data files:

```
sherpa> READ DATA example1.pha PHA, example2.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
```

This example illustrates the input of multiple data files simultaneously. The command reads `example1.dat` as dataset number 1, and `example2.dat` as dataset number 2. Again, note that the statistical errors in the PHA data files are *not* input, but the systematic errors are input.

1.65 RECORD

Controls output of parameters values and statistics to an ASCII file.

```
sherpa> RECORD [{ON | OFF}]
```

During every iteration of a fit, points in parameter space are chosen and statistics computed. The user can access information about these intermediate fits by issuing the command `RECORD ON` and retrieving the contents of the file described below. `RECORD OFF` turns off the recording. Issuing the command `RECORD` alone causes *Sherpa* to display the current setting.

Fit information is written to the ASCII file `search-<username>.dat`, located the $ASCDS_WORK_PATH directory; type "`echo $ASCDS_WORK_PATH`" to find the location on your system.

Note that successive `FIT` commands cause the file contents to be overwritten!

The fit information may also be accessed via the *Sherpa*/S-Lang module function `get_record`.

Examples:

1. Write fit parameter values and statistics to an ASCII file:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> DATA data/example1.dat
```

```
sherpa> POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> SOURCE 1 = my
sherpa> METHOD SIMPLEX
sherpa> RECORD ON
sherpa> FIT
 smplx: v1.3
 smplx:  initial statistic value =     3.25453E+02
 smplx:     converged to minimum =     3.44058E-01 at iteration =     82
 smplx:    final statistic value =     3.44058E-01
            my.c0  -0.996019
            my.c1  0.672607
            my.c2  0.923389

sherpa> $echo $ASCDS_WORK_PATH
/tmp
sherpa> $ls /tmp/search*
/tmp/search-<username>dat
sherpa> $more /tmp/search-<username>.dat
my.c0   my.c1   my.c2   Chi-Squared
33   0   0   325.453
33   0   0   325.453
32.34   0.01   0.01   312.682
32.34   0.01   0.01   312.682
31.9   0.00666667   0.00333333   303.974
...
```

1.66 REGION-PROJECTION — REG-PROJ

Creates a contour plot of confidence regions using the PROJECTION algorithm. The commands REG-PROJ and REGPROJ are abbreviated equivalents.

```
sherpa> REGION-PROJECTION [<dataset range> | ALLSETS] <arg_1> <arg_2>
```

where <dataset range> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to compute contours using data from all appropriate datasets.

The two command-line arguments may be:

Argument	Description
<sherpa_modelname>.{<paramname> | <#>}	A specified model component parameter (*e.g.*, GAUSS.pos).
<modelname>.{<paramname> | <#>}	A specified model component parameter (*e.g.*, g.pos).

Table 1.38: REGION-PROJECTION Command Arguments

The user may configure `REGION-PROJECTION` via the *Sherpa* state object structure `regproj`. The current values of the fields of this structure may be displayed using the command `print(sherpa.regproj)`, or using the more verbose *Sherpa*/S-Lang module function `list_regproj()`.

The structure fields are:

Field	Description
`fast`	If 1, use a fast optimization algorithm (`LEVENBERG-MARQUARDT` or `SIMPLEX`) regardless of the current `METHOD`. If 0, use the current `METHOD`.
`expfac`	A multiplicative factor that expands the grid limits estimated by the `COVARIANCE` algorithm, *if* the grid limits are determined automatically (see `arange`, and below).
`arange`	If 1, the grid limits are to be determined automatically. If 0, the grid limits are specified (see `min` and `max`).
`min`	An array of length two giving the grid minima for each plot axis. These are always linear quantities, regardless of the setting of `log` (see below). The array is ignored if `arange` = 1.
`max`	An array of length two giving the grid maxima for each plot axis. These are always linear quantities, regardless of the setting of `log` (see below). The array is ignored if `arange` = 1.
`log`	An array of length two specifying whether to use linear (0) or logarithmic (1) spacing of grid points along each plot axis.
`nloop`	An array of length two specifying the number of grid points along each plot axis.
`sigma`	An array of arbitrary length specifying the number of σ for each contour. The length of the array specifies the number of contours.

Table 1.39: `regproj` Structure Fields

Field values may be set using directly. If the field does not contain an array, *e.g.*,

```
sherpa> sherpa.regproj.arange = 0
```

and if it does contain an array, *e.g.*,

```
sherpa> sherpa.regproj.nloop = [25,20]
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_regproj()`.

The confidence regions are determined by varying each selected parameter's value on the determined (`arange = 1`) or specified (`arange = 0`) grid, computing the best-fit statistic at each grid point, and interpolating on the grid. `REGION-PROJECTION` differs from `REGION-UNCERTAINTY` (Section 1.67) in that all other thawed parameters are allowed to float to new best-fit values, instead of being fixed to their best-fit values. This makes `REGION-PROJECTION` contours more accurate, but causes their computation to proceed more slowly.

For a fuller theoretical description of error estimation, see `PROJECTION` (Section 1.61), `UNCERTAINTY` (Section 1.85), and `COVARIANCE` (Section 1.21).

If `arange = 1`, then the grid limits for the plot are determined automatically. For increased speed, the grid limits are determined using the error estimates from `COVARIANCE`. The covariance errors are computed assuming a change in fit statistic from the best-fit value of ΔS, which is a function of the largest value of `sigma` (*e.g.*, $\Delta S = 11.8$ if the statistic is χ^2 and 3 is the largest element of the array `sigma`. The covariance errors are then multiplied by `expfac` and are subtracted from and added to the best-fit parameter values to determine the lower and upper grid limits.

Note that the output of `COVARIANCE` may be adversely affected if the problem is not well-posed (since a matrix inversion is involved in the error calculation), or if the fit did not reach its local minimum. If the plot does not appear correct, consider running `COVARIANCE` to check the errors, and/or setting the plot limits manually.

For increased speed, the best fit at each grid point is calculated using the `LEVENBERG-MARQUARDT` optimization method (Section 4.5) if `fast = 1` and a χ^2 statistic is used to fit; the user's chosen optimization method is then reset after the plot is shown. (If `fast = 1` and a non-χ^2 statistic is used during fitting, then the `SIMPLEX` method, Section 4.5, is used instead.)

The grid-point values and best-fit statistics at each grid point may be retrieved using the *Sherpa*/S-Lang module function `get_regproj`. See the examples below.

Examples:

1. List the current and default values of the `regproj` structure, and restore the default values:

```
sherpa> sherpa.regproj.arange = 0
sherpa> sherpa.regproj.log = [1,1]
sherpa> sherpa.regproj.sigma = [1,3,5]
sherpa> list_regproj()
Parameter   Current         Default                          Description
-------------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            0               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [1,1]           [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop       [10,10]         [10,10]   Number of grid points, each axis
sigma       [1,3,5]         [1,2,3]      Number of sigma, each contour
sherpa> restore_regproj()
sherpa> list_regproj()
Parameter   Current         Default                          Description
-------------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            1               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [0,0]           [0,0]  Log-spacing: 0(n)/1(y), each axis
```

```
nloop       [10,10]         [10,10]   Number of grid points, each axis
sigma       [1,2,3]         [1,2,3]      Number of sigma, each contour
```

2. Determine 1 and 2σ confidence regions for a fit:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -10
sherpa> FIT
...
sherpa> sherpa.regproj.sigma = [1,2]
sherpa> sherpa.regproj.expfac = 4
sherpa> REGION-PROJECTION my.c0 my.c2
Region-Projection: computing grid size with covariance...done.
                   outer grid loop 20% done...
                   outer grid loop 40% done...
                   outer grid loop 60% done...
                   outer grid loop 80% done...
Minimum: 0.255419
Levels are: 2.55142 6.43642
...
```

3. Determine 1 and 2σ confidence regions for a fit using manually set grid limits:

```
sherpa> sherpa.regproj.arange = 0
sherpa> sherpa.regproj.min = [-15,0]
sherpa> sherpa.regproj.max = [15,2]
sherpa> REGION-PROJECTION my.c0 my.c2
Region-Projection: grid size set by user.
                   outer grid loop 20% done...
                   outer grid loop 40% done...
                   outer grid loop 60% done...
                   outer grid loop 80% done...
Minimum: 0.255419
Levels are: 2.55142 6.43642
```

4. Save the results of `REGION-PROJECTION` to an ASCII file:

```
[...run REGION-PROJECTION...]
sherpa> my_var = get_regproj()
sherpa> writeascii("my_output.dat",my_var.x0,my_var.x1,my_var.y)
sherpa> quit
Goodbye.
unix> more my_output.dat
-15     0       12.8958
-15     0.222222        14.7571
-15     0.444444        18.5405
...
```

1.66.1 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `REGION-PROJECTION` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `REGION-PROJECTION` may not be valid unless the parameters are at their best-fit values.

1.67 REGION-UNCERTAINTY — REG-UNC

Creates a contour plot of confidence regions using the `UNCERTAINTY` algorithm. The commands `REG-UNC` and `REGUNC` are abbreviated equivalents.

```
sherpa> REGION-UNCERTAINTY [<dataset range> | ALLSETS] <arg_1> <arg_2>
```

where <dataset range> = `#`, or more generally `#:#,#:#,...`, such that `#` specifies a dataset number, and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to compute contours using data from all appropriate datasets.

The two command-line arguments may be:

Argument	Description
<`sherpa_modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `GAUSS.pos`).
<`modelname`>.{<`paramname`> | <`#`>}	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.40: `REGION-UNCERTAINTY` Command Arguments

The user may configure `REGION-UNCERTAINTY` via the *Sherpa* state object structure `regunc`. The current values of the fields of this structure may be displayed using the command `print(sherpa.regunc)`, or using the more verbose *Sherpa*/S-Lang module function `list_regunc()`.

The structure fields are:

Field	Description
`expfac`	A multiplicative factor that expands the grid limits estimated by the `UNCERTAINTY` algorithm, *if* the grid limits are determined automatically (see `arange`, and below).
`arange`	If 1, the grid limits are to be determined automatically. If 0, the grid limits are specified (see `min` and `max`).
`min`	An array of length two giving the grid minima for each plot axis. These are always linear quantities, regardless of the setting of `log` (see below). The array is ignored if `arange` = 1.

`max`	An array of length two giving the grid maxima for each plot axis. These are always linear quantities, regardless of the setting of `log` (see below). The array is ignored if `arange` = 1.
`log`	An array of length two specifying whether to use linear (0) or logarithmic (1) spacing of grid points along each plot axis.
`nloop`	An array of length two specifying the number of grid points along each plot axis.
`sigma`	An array of arbitrary length specifying the number of σ for each contour. The length of the array specifies the number of contours.

Table 1.41: `regproj` Structure Fields

Field values may be set using directly. If the field does not contain an array, *e.g.*,

```
sherpa> sherpa.regunc.arange = 0
```

and if it does contain an array, *e.g.*,

```
sherpa> sherpa.regunc.nloop = [25,20]
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_regunc()`.

The confidence regions are determined by varying each selected parameter's value along an automatically determined grid, computing the best-fit statistic at each grid point and interpolating. `REGION-UNCERTAINTY` differs from `REGION-PROJECTION` (Section 1.66) in that all other thawed parameters are fixed to their best-fit values, rather than being allowed to float to new best-fit values. This makes `REGION-UNCERTAINTY` contours less accurate, but causes them to be computed much more quickly. For a fuller theoretical description of error estimation, see `PROJECTION` (Section 1.61), `UNCERTAINTY` (Section 1.85), and `COVARIANCE` (Section 1.21).

The grid limits for the plot are determined automatically using the `UNCERTAINTY` algorithm (Section 1.85). Each parameter's value is varied until the fit statistic is increased by ΔS, which is a function of the largest value of `sigma` (*e.g.*, $\Delta S = 11.8$ if the statistic is χ^2 and 3 is the largest element of the array `sigma`. The uncertainty errors are then multiplied by `expfac` and are subtracted from and added to the best-fit parameter values to determine the lower and upper grid limits.

The grid-point values and best-fit statistics at each grid point may be retrieved using the *Sherpa*/S-Lang module function `get_regproj`. See the examples below.

Examples:

1. List the current and default values of the `regunc` structure, and restore the default values:

```
sherpa> sherpa.regunc.arange = 0
```

```
sherpa> sherpa.regunc.log = [1,1]
sherpa> sherpa.regunc.sigma = [1,3,5]
sherpa> list_regunc()
Parameter   Current        Default                        Description
---------------------------------------------------------------------
expfac           3              3         Expansion factor for grid
arange           0              1             Auto-range: 0(n)/1(y)
min          [0,0]          [0,0]         Minimum values, each axis
max          [0,0]          [0,0]         Maximum values, each axis
log          [1,1]          [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop      [40,40]        [40,40]   Number of grid points, each axis
sigma      [1,3,5]        [1,2,3]      Number of sigma, each contour
sherpa> restore_regunc()
sherpa> list_regunc()
Parameter   Current        Default                        Description
---------------------------------------------------------------------
expfac           3              3         Expansion factor for grid
arange           1              1             Auto-range: 0(n)/1(y)
min          [0,0]          [0,0]         Minimum values, each axis
max          [0,0]          [0,0]         Maximum values, each axis
log          [0,0]          [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop      [40,40]        [40,40]   Number of grid points, each axis
sigma      [1,2,3]        [1,2,3]      Number of sigma, each contour
```

2. Determine $1, 2, 3,$ and 4σ confidence regions for a fit:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -15
sherpa> FIT
...
sherpa> sherpa.regunc.sigma = [1,2,3,4]
sherpa> REGION-UNCERTAINTY my.c0 my.c2
Region-Uncertainty: computing grid size...done.
                    outer grid loop 20% done...
                    outer grid loop 40% done...
                    outer grid loop 60% done...
                    outer grid loop 80% done...
Minimum: 0.255419
Levels are: 2.55142 6.43642 12.0854 19.5904
```

3. Determine 1 and 3σ confidence regions for the same data:

```
sherpa> sherpa.regunc.sigma = [1,3]
sherpa> REGION-UNCERTAINTY my.c0 my.c2
Region-Uncertainty: computing grid size...done.
                    outer grid loop 20% done...
                    outer grid loop 40% done...
                    outer grid loop 60% done...
```

```
                        outer grid loop 80% done...
Minimum: 0.255419
Levels are: 2.55142 12.0854
```

4. Save the results of REGION-UNCERTAINTY to an ASCII file:

```
[...run REGION-UNCERTAINTY...]
sherpa> my_var = get_regunc()
sherpa> writeascii("my_output.dat",my_var.x0,my_var.x1,my_var.y)
sherpa> quit
Goodbye.
unix> more my_output.dat
-12.7129       -0.154565       418.014
-12.7129       -0.0945145      390.736
-12.7129       -0.0344636      364.511
...
```

1.67.1 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the REGION-UNCERTAINTY command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of REGION-UNCERTAINTY may not be valid unless the parameters are at their best-fit values.

1.68 RENAME

Changes the name that has been given to a model component by the user.

```
sherpa> RENAME <modelname> <new_modelname>
```

where <**modelname**> is the name that has been given to a model component by the user, and <**new_modelname**> is the new name to be assigned.

Examples:

1. Change the name of a model component:

```
sherpa> PARAMPROMPT OFF
sherpa> GAUSS[modelb]
sherpa> RENAME modelb modelB
sherpa> SHOW modelB
gauss1d[modelB]
```

```
     Param   Type      Value        Min        Max                 Units
     -----   ----      -----        ---        ---                 -----
 1    fwhm thawed    10.0000 1.1755e-38 3.4028e+38
 2     pos thawed     0.0000-3.4028e+38 3.4028e+38
 3    ampl thawed     1.0000-3.4028e+38 3.4028e+38
```

1.69 RESET

Restores settings and/or parameter values.

```
sherpa> RESET [<arg>]
```

<arg> may be:

Argument	Description
none	Equivalent to `RESET MODELS`
`ALL`	Equivalent to issuing `RESET MODELS` *and* `RESET METHOD`
`MODELS`	Restores the parameter values of all current model components to their latest user-defined (or `GUESS` command-defined) values.
<**sherpa_model_stack**> [`#` [`ID`]]	Restores the parameter values of all components of the specified model stack (`SOURCE`, `BACKGROUND`, `PILEUP`, [B]`NOISE`, or `KERNEL`) to their latest user-defined (or `GUESS` command-defined) values. # is a dataset number (default 1); see `BACKGROUND` for an explanation of `ID`.
<**sherpa_modelname**>	Restores the parameter values of the specified model component to their latest user-defined (or `GUESS` command-defined) values.
<**modelname**>	Restores the parameter values of the specified model component to their latest user-defined (or `GUESS` command-defined) values.
`METHOD`	Restores the parameter values of the current optimization method to their initial default values.
<**sherpa_methodname**>	Restores the parameter values of the specified optimization method to their initial default values.
[B]`ERRORS` [`#` [`ID`]]	Removes user-defined source or background data error values. Note that if the errors are not redefined, then they are subsequently computed using the current choice of `statistic`. # is a dataset number (default 1); see `BERRORS` for an explanation of `ID`.
[B]`SYSERRORS` [`#` [`ID`]]	Resets the systematic error for every source or background data point to 0. # is a dataset number (default 1); see `BSYSERRORS` for an explanation of `ID`.

`FILTER [#]`	Clears user-defined filters. # is a dataset number (default 1). *NOTE:* the implementation of this command in CIAO 3.0 contains a bug, such that sometimes the source and background filters are reset, and sometime only one or the other; also, there is no `RESET BFILTER`.
`[B]WEIGHTS [# [ID]]`	Resets the statistical weight assignment for every source or background data point to 1. Removes user-defined source or background statistic weight settings. # is a dataset number (default 1); see `READ BWEIGHTS` for an explanation of `ID`.

Table 1.42: `RESET` Command Arguments

`ERASE` is a related command that may be used to remove all user inputs and user-defined settings, or to remove a model component from the current *Sherpa* session.

Examples:

1. Restore the parameter values of the current method to the initial values:

```
sherpa> METHOD GRID
sherpa> SHOW METHOD
Optimization Method: Grid

      Name       Value         Min         Max                  Description
      ----       -----         ---         ---                  -----------
 1  totdim           4           1          24       Number of free parameters
 2 nloop01          10           1       1e+07           Number of grid points
 3 nloop02          10           1       1e+07           Number of grid points
 4 nloop03          10           1       1e+07           Number of grid points
 5 nloop04          10           1       1e+07           Number of grid points

sherpa> GRID.nloop01 = 2000
sherpa> SHOW METHOD
Optimization Method: Grid

      Name       Value         Min         Max                  Description
      ----       -----         ---         ---                  -----------
 1  totdim           4           1          24       Number of free parameters
 2 nloop01        2000           1       1e+07           Number of grid points
 3 nloop02          10           1       1e+07           Number of grid points
 4 nloop03          10           1       1e+07           Number of grid points
 5 nloop04          10           1       1e+07           Number of grid points

sherpa> RESET METHOD
sherpa> SHOW METHOD
Optimization Method: Grid

      Name       Value         Min         Max                  Description
```

```
     ----       -----        ---         ---                   -----------
  1  totdim         4          1          24       Number of free parameters
  2 nloop01        10          1      1e+07            Number of grid points
  3 nloop02        10          1      1e+07            Number of grid points
  4 nloop03        10          1      1e+07            Number of grid points
  5 nloop04        10          1      1e+07            Number of grid points
```

2. Restore the parameter values, of the current model components, to the latest user-defined values:

```
sherpa> DATA example.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GAUSS[modelb]
sherpa> SOURCE 1 = modelb
sherpa> modelb.pos = 1.0
sherpa> SHOW modelb
gauss1d[modelb]  (integration: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     2.0185     0.0202   201.8513
 2    pos thawed          1          1          4
 3   ampl thawed         17     0.1700       1700
sherpa> FIT
sherpa> SHOW modelb
gauss1d[modelb]  (integration: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     3.3671     0.0202   201.8513
 2    pos thawed          4          1          4
 3   ampl thawed    10.1912     0.1700       1700
sherpa> RESET modelb
sherpa> SHOW modelb
gauss1d[modelb]  (integration: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     2.0185     0.0202   201.8513
 2    pos thawed          1          1          4
 3   ampl thawed         17     0.1700       1700
```

1.70 RESPONSE — INSTRUMENT

Defines the instrument model to be used.

See the `INSTRUMENT` command (Section 1.46) for information and examples.

1.71 RUN — FIT

Initiates fitting.

See the `FIT` command (Section 1.35) for further information.

1.72 SAVE

Saves information to an ASCII file.

```
sherpa> SAVE <arg> <filename>
```

<**arg**> may be:

Argument	**Description**
`ALL`	Equivalent to issuing the commands `SAVE BACKGROUND`, `SAVE INSTRUMENT`, `SAVE METHOD`, `SAVE SYSTEM`, `SAVE SOURCE`, and `SAVE STATISTIC` commands, in addition to saving the information needed to fully restore a *Sherpa* session, such as the names of the loaded datasets, and the currently defined filters.
`SYSTEM`	Saves the parameter values and ranges for all current source, background, and instrument model components.
`MODELS`	Saves current model component names (*e.g.*, `g` for the definition `GAUSS`[`g`]).
<**sherpa_model_stack**> [`#` [`ID`]]	Saves current names for all model components of the specified model stack (`SOURCE`, `BACKGROUND`, `PILEUP`, [`B`]`NOISE`, or `KERNEL`; also [`B`]`INSTRUMENT`) to their latest user-defined values. `#` is a dataset number (default 1); see `BACKGROUND` for an explanation of `ID`.
<**model_stack**>	Saves current names for all model components of the specified user-defined model stack.
<**sherpa_modelname**>	Saves the parameter values and ranges for the specified source, background, or instrument model component.
<**modelname**>	Saves the parameter values and ranges for the specified source, background, or instrument model component.
`METHOD`	Saves the current parameter values of the selected optimization method.
<**sherpa_methodname**>	Saves the current parameter values of the specified optimization method.
`STATISTIC`	Saves the name of the current statistic.

Table 1.43: `SAVE` Command Arguments

The `SAVE` command may be issued at any time. Each time the command is issued the specified ASCII file <`filename`> is written, saving the specified settings and values. If the specified ASCII file already exists, it will be overwritten.

Tip: the command `SAVE ALL` <`filename`> is useful for saving the current state of the user's *Sherpa* session; the current session may be restored at a later time by using <`filename`> as a *Sherpa* script.

Examples:

1. Write all settings, definitions, filters, and parameter values to an ASCII file:

```
sherpa> DATA data/example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example.rmf
ARF is being input from:
  <directory_path>/example.arf
Background data are being input from:
  <directory_path>/example_bkg.pha
sherpa> LPLOT DATA
sherpa> IGNORE ALL
sherpa> NOTICE PLOT
sherpa> BBODY[modelh]
modelh.space parameter value [0]
modelh.kT parameter value [0.592333]
modelh.ampl parameter value [0.000464907]
sherpa> SOURCE = modelh
sherpa> SAVE ALL mysession1.shp
sherpa> EXIT
Goodbye.
```

In this example, the ASCII file named `mysession1.shp` is written with: all program settings; all parameter values for the current method; all parameter values, including ranges and `delta` settings, for the current model components; source definitions for all datasets; and the name of the current statistic. Note that information about an interactively-specified filter is also included.

2. Restore a session using the results of a previous `SAVE ALL` command:

```
sherpa> USE mysession1.shp
```

The command `USE mysession1.shp` restores the example session above, using the file `mysession1.shp`.

3. Write all parameter values to an ASCII file:

```
sherpa> SAVE modelh mymodelh.shp
sherpa> $more mymodelh.shp
modelh integrate on
modelh.space.min = 0
modelh.space.max = 1
modelh.space.value = 0
modelh.space.type = freeze
modelh.kT.min = 0.00592333
modelh.kT.max = 59.2333
modelh.kT.value = 0.592333
modelh.kT.type = thaw
modelh.ampl.min = 4.64907e-06
modelh.ampl.max = 0.0464907
modelh.ampl.value = 0.000464907
modelh.ampl.type = thaw
```

In this example, the ASCII file named `mymodelh.shp` is written with the parameter values, including parameter ranges, for `modelh`.

1.73 SEARCHMETHOD — METHOD

Specifies the optimization method.

See the `METHOD` command (Section 1.53) for the <**sherpa_methodname**> definition and examples.

1.74 SETBACK

Sets attributes of a background dataset.

```
sherpa> SETBACK [# [ID]] <arg> = <numeric>
```

where `#` specifies the number of the background dataset (default dataset number is 1). The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The argument <**arg**> is one of the following options:

Argument	Description
BACKSCALE	A normalizing quantity which can indicate the the ratio of the area of the background extraction region in an image to the full image area.
TIME	The exposure time of the background observation.

Table 1.44: `SETBACK` Command Arguments

The primary use of this command is to set the attributes of non-PHA datasets (*e.g.*, ASCII datasets). Setting `TIME` affects the normalization of the background model, which is entered with the `BACKGROUND` command (Section 1.9).

Setting `BACKSCAL` affects the relative normalization of the background model when it is applied to a source region. For instance, if the background model amplitude in a background dataset bin is M_B, the `BACKSCAL` of that dataset is B_B, and the `BACKSCAL` of the source dataset is B_S, then the contribution of the background to the source region spectrum is

$$M'_B = \frac{B_S}{B_B} M_B \tag{1.4}$$

For related information, see `SETDATA` (Section 1.75).

1.75 SETDATA

Sets attributes of a source dataset.

```
sherpa> SETDATA [#] <arg> = <numeric>
```

where # specifies the number of the background dataset (default dataset number is 1).

The argument <**arg**> is one of the following options:

Argument	**Description**
`BACKSCALE`	A normalizing quantity which can indicate the the ratio of the area of the source extraction region in an image to the full image area.
`TIME`	The exposure time of the source observation.

Table 1.45: `SETDATA` Command Arguments

The primary use of this command is to set the attributes of non-PHA datasets (*e.g.* ASCII datasets). Setting `TIME` affects the normalization of the source model, which is entered with the `SOURCE` command (Section 1.77).

Setting `BACKSCAL` affects the relative normalization of the background model when it is applied to a source region. For instance, if the background model amplitude in a background dataset bin is M_B, the `BACKSCAL` of that dataset is B_B, and the `BACKSCAL` of the source dataset is B_S, then the contribution of the background to the source region spectrum is

$$M'_B = \frac{B_S}{B_B} M_B \tag{1.5}$$

For related information, see `SETBACK` (Section 1.74).

Examples:

1. Set the time of an input ASCII dataset:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLY[p]
sherpa> THAW p.c1 p.c2
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 325.453
 LVMQT: final statistic value = 0.255412 at iteration 2
             p.c0  0.305218
             p.c1  -0.142263
             p.c2  1.01643

sherpa> SETDATA TIME = 100
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 1.43934e+06
 LVMQT: final statistic value = 0.255412 at iteration 3
             p.c0  0.00292297
             p.c1  -0.00133705
             p.c2  0.0101542
```

2. Set the times and backscales for an input dataset with mean amplitude 60 counts and a background dataset with mean amplitude 20 counts:

```
sherpa> DATA spec.dat
sherpa> BACK back.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = CONST[co]
sherpa> BG = CONST[bo]
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 980.167
 LVMQT: final statistic value = 142.424 at iteration 3
            co.c0  39.3331
            bo.c0  18.4647

sherpa> SETDATA TIME = 10
sherpa> SETDATA BACKSCALE = .1
sherpa> SETBACK TIME = 100
sherpa> SETBACK BACKSCALE = .25
sherpa> bo.c0.min = 0
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 7.34745e+07
 LVMQT: final statistic value = 142.424 at iteration 3
            co.c0  5.70591
            bo.c0  0.0738588
```

1.76 SHOW

Reports current status.

```
sherpa> SHOW [<arg>]
```

<**arg**> may be:

Argument	**Performs the following:**
[ALL]	Reports all current settings and parameter values. The commands SHOW and SHOW ALL are equivalent.
VERBOSE	Reports all current settings and parameter values, in verbose mode.
MODELS	Displays information about all current model stacks and model component parameter values.
<**sherpa_model_stack**> [# [ID]]	Displays the parameter values of all components of the specified model stack (SOURCE, BACKGROUND, PILEUP, [B]NOISE, or KERNEL). # is a dataset number (default 1); see BACKGROUND for an explanation of ID.
<**model_stack**>	Displays the parameter values of all model components of the specified user-defined model stack.
<**sherpa_modelname**>	Displays the parameter values of the specified model component.
<**modelname**>	Displays the parameter values of the specified model component.
<**sherpa_plot_quantity**>	Lists the specified quantity; this includes everything that can be plotted/imaged with *Sherpa* except fits, unconvolved data/fits, unconvolved model amplitudes, and model stacks. See, *e.g.*, documentation on the command LPLOT. (This option of SHOW may be deprecated in future versions of *Sherpa* since it largely duplicates the WRITE command.
METHOD	Displays the name of the current optimization method, and lists parameter values.
<**sherpa_methodname**>	Displays the name of the specified optimization method, and lists parameter values.
STATISTIC	Displays the name of the current statistic.
FAKEIT	Displays the exposure time and backscale to be used for FAKEIT.
PILEUP	Reports on the pileup fractions from the most recent fit. See PILEUP for more details.

Table 1.46: SHOW Command Arguments:

Note: The behavior of the SHOW command is controlled by the environment variable $PAGER. The default setting for this variable should be:

```
unix% echo $PAGER
/bin/more
```

This setting causes the results of the SHOW command to scroll down the screen. However, the user may prefer to use:

```
unix% setenv PAGER "/bin/less -s"
unix% echo $PAGER
/bin/less -s
```

Note: The SHOW command can generate more than one screen of information, particularly after data have been read, filters and models defined, models fit to the data, and statistics generated. A : symbol indicates additional information is on the next screen; tapping the space bar moves the pointer to the next screen. The end of SHOW output is indicated by END; tapping q quits the SHOW interface. (However, there are many more options at this point, including saving the screen output to a file. Tapping h (for help) will cause information on these options to be displayed.

Note: The commands SHOW MODELS, SHOW <modelname>, and SHOW SOURCE currently do not return information about model parameter delta settings. Information about model parameter delta settings is included when using SAVE.

Examples:

1. Report all current settings:

```
sherpa> SHOW

Optimization Method: Levenberg-Marquardt
Statistic:           Chi-Squared Gehrels
```

   This example illustrates all of the initial *Sherpa* settings; since no models have been defined, none are shown.

2. Report on the model components of different source models:

```
sherpa> READ DATA 1 example1.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POLY[modela]
sherpa> GAUSS[modelb]
sherpa> SOURCE 1 = modela
sherpa> SOURCE 2 = modelb
sherpa> SHOW SOURCE 1
```

```
Source 1: modela
polynom1d[modela]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1     c0 thawed         33         -1         65
 2     c1 frozen          0  -914.2857   914.2857
 3     c2 frozen          0  -130.6122   130.6122
 4     c3 frozen          0         -1         65
 5     c4 frozen          0         -1         65
 6     c5 frozen          0         -1         65
 7     c6 frozen          0         -1         65
 8     c7 frozen          0         -1         65
 9     c8 frozen          0         -1         65
10 offset frozen          0         -1          8
sherpa> SHOW SOURCE 2
Source 2: modelb
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     5.1306 5.1306e-02   513.0582
 2    pos thawed          8          1          8
 3   ampl thawed         65       0.65       6500
```

1.77 SOURCE — SRC

Defines the source model expression to be used for fitting a dataset. The command `SRC` is an abbreviated equivalent.

```
sherpa> SOURCE [<dataset range> | ALLSETS] = <modelExpr>
```

<`dataset range`> = # (or more generally #:#,#:#, *etc.*) such that # specifies a dataset number and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The model expression, <`modelExpr`>, is an algebraic combination of one or more of the following elements:

```
{<sherpa_modelname> | <sherpa_modelname>[modelname] |
 <modelname> | <model_stack> | <nested_model>}
```

along with numerical values. The following operators are recognized: + - * / ( ) { }. See the `CREATE` (Section 1.23) command for further information.

Note that:

- The documentation on *Sherpa Models* (Section 5) contains a summary list, and descriptions, of the models that are available within *Sherpa*, which include models from *XSPEC*, v. 11.3.

- By default, if the model expression includes a model component that has not previously been established, *Sherpa* will prompt for the initial parameter values for that model component. This prompting can be turned off using the `PARAMPROMPT OFF` (Section 1.58) command.

To reset a source model stack, issue the command:

```
sherpa> SOURCE [<dataset range> | ALLSETS] =
```

Examples:

1. Establish a model component, and define it as the source model to be used for fitting a specific dataset:

   ```
   sherpa> SOURCE 2 = GAUSS
   GAUSS.fwhm parameter value [10]
   GAUSS.pos parameter value [0]
   GAUSS.ampl parameter value [1]
   sherpa>
   ```

   This command defines the *Sherpa* model `GAUSS` as the source model to be used for fitting dataset number 2. Note that the user accepted the given initial values for all of the parameters, using the <RETURN> key.

2. Establish a model component and assign it a name; define the model component as the source model to be used for fitting a specific dataset:

   ```
   sherpa> PARAMPROMPT OFF
   Model parameter prompting is off
   sherpa> GAUSS[modelb]
   sherpa> SOURCE 2 = modelb
   ```

   In the first command, the name `modelb` is given to the *Sherpa* model component `GAUSS` (see the `CREATE` command for more information on the model language syntax). The second command defines this model as the source model to be used for fitting dataset number 2.

3. Establish model components, and assign them names; create a source model expression to be used for fitting a specific dataset:

   ```
   sherpa> POW[modelc]
   sherpa> GAUSS[modelf]
   sherpa> SOURCE 1 = modelc + modelf
   ```

   The last command in this series assigns the model expression `modelc + modelf` as the source model to be used for fitting dataset number 1. The following commands each assign various other model expressions to source models for dataset numbers 2, 3, and 4 respectively:

   ```
   sherpa> SOURCE 2 = 10*(modelc + modelf)
   sherpa> SOURCE 3 = (modelc - modelf)/2
   sherpa> SOURCE 4 = 0.5*modelc + 0.7*modelf
   ```

See the CREATE command for further information about creating model expressions.

4. Establish a model component, assign it a name, and define it as the source model to be used for fitting:

```
sherpa> ERASE ALL
sherpa> SOURCE 1 = POW[modelc]
```

In this single command, the name modelc is assigned to the *Sherpa* model component POW, and then this model is defined as the source model to be used for fitting dataset number 1.

5. Establish model components, assign them names, and use them to define a source model expression to be used for fitting:

```
sherpa> ERASE ALL
sherpa> SOURCE 2 = GAUSS[modelb] + POW[modelc]
```

In this single command, the following is performed: the name modelb is assigned to the *Sherpa* model component GAUSS; the name modelc is assigned to the *Sherpa* model component POW; a model expression, which here is the sum of these two models components, is defined as the source model to be used for fitting dataset number 2.

6. Establish a model component, assign it a name, define the parameters, and define it as the source model to be used for fitting:

```
sherpa> ERASE ALL
sherpa> SOURCE = POLY[modela](3.0:1.0:4.0)
```

With this single command, the name modela is assigned to the *Sherpa* model component POLY, the value of 3.0 is given to the model's first parameter (in this case parameter c0), the minimum of 1.0 is set for this parameter, the maximum of 4.0 is set for this parameter, and then this model is defined as the source model to be used for fitting dataset number 1.

7. Establish multiple model components, assign them names, and use them in a source model expression definition:

```
sherpa> SOURCE 2 = GAUSS[modelb](3:2.5:4.403, 1:-10:10, 1:-3.5:3.5)
                   + POW[modelc]
```

With this single command, the following is performed: the name modelb is assigned to the *Sherpa* model component GAUSS; various parameter values and ranges are set for the parameters of modelb; the name modelc is assigned to the *Sherpa* model component POW; a model expression, which is the sum of these two models, is defined as the source model to be used for fitting dataset number 2.

8. Establish continuum and line model stacks, and combine these stacks into a source model expression definition:

```
sherpa> PARAMPROMPT OFF
sherpa> CONT = POWLAW1D[modeld]
sherpa> ELINE = NGAUSS[modele]
sherpa> SOURCE = CONT + ELINE
```

In the second and third commands, the names modeld and modele are assigned to the *Sherpa* model components POWLAW1D and NGAUSS respectively; these model components are then assigned to the user-defined model stacks CONT and ELINE. These model stacks are then assigned to the source model expression.

1.78 SPLOT

Causes the specified 2-D data to be displayed, with a surface plot, via *ChIPS*.

```
sherpa> SPLOT [<num_plots>] <arg_1> [# [ID]] [<arg_2> [# [ID]] ...]
```

<**num_plots**> specifies the number of plotting windows to open within the *ChIPS* pane (default 1); that number sets the number of subsequent arguments. For each subsequent argument, `#` specifies the number of the dataset (default dataset number is 1), and the `ID` modifier is used for displaying background datasets, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The argument <**arg_n**> may be any of the following:

Argument	**Displays**
{{DATA | DCOUNTS} | {BACK | BDCOUNTS}}	The source|background data values
{ERRORS | BERRORS}	The estimated total errors for the source|background data values
{SYSERRORS | BSYSERRORS}	The assigned systematic errors for the source|background data values
{STATERRORS | BSTATERRORS}	The estimated statistical errors for the source|background data values
{{MODEL | MCOUNTS} | {BMODEL | BMCOUNTS}}	The (convolved) source|background model amplitudes
{DELCHI | BDELCHI}	The sigma residuals of the source|background fit: (data - model)/error
{RESIDUALS | BRESIDUALS}	The absolute residuals of the source|background fit: (data - model)
{RATIO | BRATIO}	The ratio (data/model) for source|background
{CHI SQU | BCHISQU}	The contributions to the χ^2 statistic from each source|background data point
{STATISTIC | BSTATISTIC}	The contributions to the current statistic from each source|background data point
{WEIGHT | BWEIGHT}	The statistic weight value assigned to each source|background data point
{FILTER | BFILTER}	The mask value (0|1) for each source|background data point
<**sherpa_model_stack**>	The (unconvolved) model amplitudes for the specified model stack (SOURCE, {BACKGROUND|BG}, or [B]NOISE)
<**model_stack**>	The (unconvolved) model amplitudes for the specified user-defined model stack
<**sherpa_modelname**>	The (unconvolved) amplitudes of the specified model component (*e.g.*, GAUSS2D)
<**modelname**>	The (unconvolved) amplitudes of the specified model component (*e.g.*, g)
{EXPMAP | BEXPMAP}	The unfiltered source|background 2-D EXPMAP file contents

{PSF | BPSF}	The unfiltered source|background 2-D PSF file contents

Table 1.47: `SPLOT` Command Arguments

If there is no open plotting window when an `SPLOT` command is given, one will be created automatically.

If one issues the `SPLOT` following filtering, note the following: arbitrarily filtered data cannot be passed from *Sherpa* to *ChIPS* for display; the data grid must be rectangular. Therefore, surface plots are created in three steps: (1) the smallest possible rectangle is drawn around the noticed data; (2) within this rectangle, the <`data to image`> is tranformed to <`data to image`> `* filter`; and (3) these transformed data are sent off to *ChIPS* for display.

The appearance of plots generated with this command can be changed by modifying the fields of certain state objects. See the ahelp for *Sherpa* or for sherpa.plot for more information.

NOTE: all *ChIPS* commands may be used from within *Sherpa* to modify plot characteristics. In order to view these changes, the `REDRAW` command must be issued.

See the display chapter (Chapter 2) for more information regarding data display capabilities, including modifying various plot characteristics.

Examples:

1. Display 2-D data with a surface plot:

   ```
   sherpa> DATA 3 example2Da.dat ASCII 1 2 3
   sherpa> SPLOT DATA 3
   ```

   The `SPLOT` command plots dataset number 3 as a surface plot. Dataset number 3 must be a 2-D dataset.

2. Display 2-D datasets with surface plots in multiple windows:

   ```
   sherpa> DATA 1 example2Db.dat ASCII 1 2 3
   sherpa> SPLOT 2 DATA 1 DATA 3
   ```

   This command displays a surface plot of dataset number 1 (`example2Db.dat`) in the first window, and a surface plot of dataset number 3 (`example2Da.dat`) in the second window.

1.79 STATISTIC

Specifies the fitting statistic.

```
sherpa> STATISTIC <sherpa_statisticname>
```

<`sherpa_statisticname`> is the name of one of the supported fitting statistics:

<sherpa_statisticname>	**Description**
BAYES	A Bayesian maximum likelihood function.
CASH	A maximum likelihood function.
{CHI CVAR | CHI PARENT}	χ^2 statistic with constant variance computed from the counts data.
CHI DVAR	χ^2 statistic with variance computed from the data.
CHI GEHRELS	χ^2 statistic with the Gehrels variance function.
CHI MVAR	χ^2 statistic with variance computed from model amplitudes.
CHI PRIMINI	χ^2 statistic with Primini variance function.
CSTAT	A maximum likelihood function: *XSPEC* implementation of CASH
USERSTAT	User-implemented statistic.

Table 1.48: STATISTIC Command Arguments

The default statistic is CHI GEHRELS. It is the default because of its behavior in the low-counts regime.

The statistic that has been set in the current *Sherpa* session may be listed with the command SHOW STATISTIC.

Examples:

1. Specify a fitting statistic to be used; report on it:

```
sherpa> STATISTIC BAYES
sherpa> SHOW STATISTIC
Statistic: Bayes
```

1.80 SUBTRACT

Performs background subtraction.

```
sherpa> SUBTRACT [<dataset range> | ALLSETS]
```

where <dataset range> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The SUBTRACT performs background subtraction on a channel-by-channel basis:

$$S_i' \ = \ S_i - \beta_S t_S \left[\frac{B_i}{\beta_B t_B} \right] , \tag{1.6}$$

where S_i is the source datum in bin i, B_i is the background datum in bin i, t_S and t_B are the source and background exposure times respectively, and β_S and β_B are the source and background "backscales" (*i.e.*, the value of the BACKSCAL header keyword value in a PHA file) respectively. BACKSCAL is typically defined as the ratio of data extraction area to total detector area. This equation is easily generalized if there are multiple background datasets per source dataset.

The `SUBTRACT` command may only be used when:

- the input source and background datasets are of the same size; or
- the source dataset is grouped and the background dataset is ungrouped; *Sherpa* will automatically group the background dataset to match the source dataset before doing the subtraction.

Unless the user specifies errors for the background-subtracted data, the errors are computed automatically by propagating the errors for source and background data.

To undo background subtraction, use the command `UNSUBTRACT`.

Alternative means of subtracting and unsubtracting involve using the *Sherpa*/S-Lang module functions `set_subtract` and `set_unsubtract`.

Note that while *XSPEC* automatically performs background subtraction, in *Sherpa* one must directly issue the command `SUBTRACT`.

Examples:

1. Read in source and background data, then subtract.

```
sherpa> DATA example2.pha
...
Background data are being input from:
  <directory_path>/example2_bkg.pha
sherpa> PLOTY COUNTS
sherpa> WRITE DATA
Write X-Axis: Energy (keV)  Y-Axis: Flux (Counts)
...
     0.4286        13
     0.4573        15
     0.5002        25
     0.5575        21
...
sherpa> SUBTRACT
Write X-Axis: Energy (keV)  Y-Axis: Flux (Counts)
...
     0.4286     5.2974
     0.4573     6.9893
     0.5002    10.8273
     0.5575     6.2111
...
sherpa> UNSUBTRACT
Write X-Axis: Energy (keV)  Y-Axis: Flux (Counts)
...
```

```
        0.4286         13
        0.4573         15
        0.5002         25
        0.5575         21
    ...
```

1.81 STATERRORS

Defines an expression or file to be used to specify the *statistical* errors for source data.

```
sherpa> [B]STATERRORS [<dataset range> | ALLSETS] = <errorExpr>
```

`STATERRORS` is used for specifying statistical errors in source datasets, while `BSTATERRORS` is used for specifying statistical errors in background datasets.

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The error expression, <errorExpr>, may be composed of one or more (algebraically-combined) of the following elements:

Component:	Description:
`DATA`	An input dataset
numericals	Numerical values
operators	+ - * / ( )

A few things to note:

- Statistical error expressions cannot be defined without first reading in a source dataset.
- Statistical errors are added in quadrature with systematic errors (if specified) to produce an overall error estimate in each data bin.
- Currently, statistical errors *cannot* be specified in an input file.

For additional information, see the related command `ERRORS` (Section 1.30).

Examples:

1. Define an expression to be used for the statistical errors.

   ```
   sherpa> DATA data.dat
   sherpa> STATERRORS = 5
   ```

1.82 SYSERRORS

Defines an expression or file to be used to specify the *systematic* errors for source data.

```
sherpa> SYSERRORS [<dataset range> | ALLSETS] = <errorExpr>
```

`SYSERRORS` is used for specifying statistical errors in source datasets, while `BSYSERRORS` is used for specifying statistical errors in background datasets ("ahelp bsyserrors").

<dataset range> = `#` (or more generally `#:#,#:#,` *etc.*) such that `#` specifies a dataset number and `#:#` represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The error expression, <errorExpr>, may be composed of one or more (algebraically-combined) of the following elements:

Component:	Description:
`DATA`	An input dataset
numericals	Numerical values
operators	+ - * / ( )

A few things to note:

- Systematic error expressions cannot be defined without first reading in a source dataset.
- The input systematic error for a given bin is first multiplied by the datum in that bin; the resulting product is added in quadrature with the statistical error to yield the overall error for that bin.
- The systematic errors can be specified in an input file; see `READ SYSERRORS` (Section 1.64) for more information.
- If the systematic errors are not defined using the `SYSERRORS` command or input using `READ SYSERRORS`, they are assumed to be zero.

For additional information, see the related command `ERRORS` (Section 1.30).

The systematic errors are accessible to the *Sherpa*/S-Lang module user via the functions `get_syserrors` and `set_syserrors`.

Examples:

1. Define an expression to be used for the systematic errors. In each bin, they will be computed as 0.1 times the datum.

```
sherpa> DATA data.dat
sherpa> SYSERRORS = 0.1
```

1.83 THAW

Allows model parameter(s) to vary.

```
sherpa> THAW <arg_1> [<arg_2> ...]
```

<arg> may be:

Argument	Description
`<sherpa_modelname>.{<paramname> | <#>}`	Thaws the specified model component parameter.
`<modelname>.{<paramname> | <#>}`	Thaws the specified model component parameter.
`<sherpa_modelname>`	Thaws all parameters of the specified model component.
`<modelname>`	Thaws all parameters of the specified model component.
`<sherpa_model_stack> [# [ID]]`	Thaws the parameters of all model components within the specified model stack (`SOURCE`, `BACKGROUND`, `PILEUP`, `[B]NOISE`, or `KERNEL`). `#` is a dataset number (default 1); see `BACKGROUND` for an explanation of `ID`.
`<model_stack>`	Thaws the parameters of all model components within the user-defined model stack.

Table 1.51: `THAW` Command Arguments

The command `FREEZE` (Section 1.37) is used to prohibit model parameter values from varying.

In addition, model parameters may be thawed using the equivalent command

- `sherpa> <modelname>.{<paramname>|<#>}.TYPE = THAW`

Model parameters may also be thawed using the *Sherpa*/S-Lang module functions `set_thawed` and `set_par`.

Examples:

1. Thaw a model parameter:

   ```
   sherpa> GAUSS[modelb]
   modelb.fwhm parameter value [10]
   modelb.pos parameter value [0]
   modelb.ampl parameter value [1]
   sherpa> THAW modelb.ampl
   ```

   The final command thaws the parameter `ampl` of `modelb`.

2. Freeze a model parameter and thaw a model parameter:

```
sherpa> FREEZE modelb.3
sherpa> THAW modelb.3
```

First, the third parameter of `modelb` is frozen. The last command then thaws the third parameter of `modelb`.

3. Freeze all model parameters and thaw two model parameters:

```
sherpa> FREEZE modelb
sherpa> THAW modelb.fwhm modelb.pos
```

First, all parameters of `modelb` are frozen. The last command then thaws the `fwhm` and `pos` parameters of `modelb`.

4. Freeze or thaw all source component parameters at once:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
sherpa> GAUSS[modelf]
sherpa> SOURCE 2 = modelc + modelf
sherpa> FREEZE SOURCE 2
sherpa> SHOW SOURCE 2
Source 2:

(modelc + modelf)
powlaw1d[modelc]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
 1  gamma frozen          1        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl frozen          1      1e-20 3.4028e+38
gauss1d[modelf]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
 1   fwhm frozen         10 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl frozen          1-3.4028e+38 3.4028e+38
sherpa> THAW SOURCE 2
sherpa> SHOW SOURCE 2
(modelc + modelf)
powlaw1d[modelc]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
    -----   ----      -----       ---        ---                 -----
 1  gamma thawed          1        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1      1e-20 3.4028e+38
gauss1d[modelf]  (integrate: on)
    Param   Type      Value       Min        Max                 Units
```

```
    -----   ----      -----        ---        ---                   -----
 1   fwhm thawed         10 1.1755e-38 3.4028e+38
 2    pos thawed          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
```

This example illustrates the use of `FREEZE SOURCE` and `THAW SOURCE` to freeze and thaw all source component parameters at once, respectively. Note that thawing of some model parameters (*e.g.*, `POWLAW1D.ref`) is not permitted.

5. Thaw a model parameter:

```
sherpa> modelb.ampl.TYPE = THAW
```

This command thaws the parameter `ampl` of `modelb`. The following commands are each equivalent:

```
sherpa> modelb.3.TYPE = THAW
sherpa> THAW modelb.ampl
sherpa> THAW modelb.3
```

6. Freeze all model parameters at once and thaw all source parameters at once:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> DATA data/example.pha
sherpa> GAUSS[modelc]
sherpa> SOURCE = modelb + modelc
sherpa> FREEZE modelc
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                   Units
    -----   ----      -----        ---        ---                   -----
 1   fwhm frozen          2 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                   Units
    -----   ----      -----        ---        ---                   -----
 1   fwhm frozen     0.7113     0.0071    71.1283
 2    pos frozen     0.9442     0.0276    14.5494
 3   ampl frozen     0.0001 1.0564e-06     0.0106
sherpa> THAW SOURCE
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                   Units
    -----   ----      -----        ---        ---                   -----
 1   fwhm thawed          2 1.1755e-38 3.4028e+38
 2    pos thawed          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
```

```
    Param   Type      Value        Min        Max                Units
    -----   ----      -----        ---        ---                -----
 1   fwhm thawed     0.7113     0.0071    71.1283
 2    pos thawed     0.9442     0.0276    14.5494
 3   ampl thawed     0.0001 1.0564e-06     0.0106
```

Note that the command `FREEZE modelc` freezes all parameters of the source model component `modelc`, while `THAW SOURCE` thaws all parameters of both source model components.

1.84 TRUNCATE

Resets negative model amplitudes to zero.

```
sherpa> TRUNCATE [<dataset range> | ALLSETS] [{ON | OFF}]
```

where <`dataset range`> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset number 1.

The command `TRUNCATE` resets any negative model amplitudes to zero (actually, 10^{-25}); truncation is necessary for the Poisson-likelihood-based statistics of *Sherpa* (`BAYES`, `CASH`, and `CSTAT`) to work properly (none of these statistics allows negative model amplitudes). Note that:

- if neither `ON` nor `OFF` is specified, the truncation setting for the specified dataset(s) is displayed;
- truncation is automatically turned on if the assigned `STATISTIC` is `BAYES`, `CASH`, or `CSTAT`; and
- truncation is never automatically turned off.

Examples:

1. Fitting with a linear model, using the Cash statistic:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1
sherpa> my.c0.min = -10
sherpa> METHOD SIMPLEX
sherpa> STATISTIC CASH
sherpa> TRUNCATE OFF
sherpa> FIT
 smplx: v1.3
 smplx:  initial statistic value =    -9.05568E+02
```

```
Error: predicted data are zero or negative.
  --   Apply commands RESET and TRUNCATE ON, and refit.
sherpa> RESET
sherpa> TRUNCATE
Model Truncation for Dataset 1: off
sherpa> TRUNCATE ON
sherpa> FIT
 smplx: v1.3
 smplx:  initial statistic value =    -9.05568E+02
 smplx:     converged to minimum =    -1.05814E+03 at iteration =     67
 smplx:    final statistic value =    -1.05814E+03
           my.c0  -6.61366
           my.c1  7.16072
```

1.85 UNCERTAINTY

Estimates confidence intervals for selected *thawed* parameters.

```
sherpa> UNCERTAINTY [<dataset range> | ALLSETS] [ <arg_1> , ... ]
```

where <dataset range> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default is to estimate limits using data from all appropriate datasets.

The command-line arguments <arg_n> may be:

Argument	Description
<sherpa_modelname>.{<paramname> | <#>}	A specified model component parameter (*e.g.*, GAUSS.pos).
<modelname>.{<paramname> | <#>}	A specified model component parameter (*e.g.*, g.pos).

Table 1.52: UNCERTAINTY Command Arguments

The user may configure UNCERTAINTY via the *Sherpa* state object structure unc. The current values of the fields of this structure may be displayed using the command print(sherpa.unc), or using the more verbose *Sherpa*/S-Lang module function list_unc().

The structure fields are:

Field	Description
sigma	Specifies the number of σ (*i.e.*, the change in statistic).
eps	The tolerance for sigma, influencing the numerical accuracy of the errorbars. (Decreasing eps increases errorbar accuracy.)

`remin`	If in the course of computing the interval, a statistic value is found that is less than the previous best-fit by more than `remin`, a new fit will be started; after minimization, `UNCERTAINTY` will run to completion.

Table 1.53: `unc` Structure Fields

Field values may be set using directly, *e.g.*,

```
sherpa> sherpa.unc.sigma = 2.6
```

NOTE: strict checking of value inputs is not done, *i.e.*, the user can errantly change arrays to scalars, *etc.* To restore the default settings of the structure at any time, use the *Sherpa*/S-Lang module function `restore_unc()`.

Confidence interval bounds are determined for each parameter in turn. A given parameter's value is varied while the values of all the other nominally thawed parameters are held fixed to their best-fit values. This is a simplistic method of estimating confidence interval bounds that gives truly accurate results only in special cases, as explained below.

Because `UNCERTAINTY` estimates confidence intervals for each parameter independently, the relationship between `UNCERTAINTY.sigma` and the change in statistic value ΔS can be particularly simple: $\sigma = \sqrt{\Delta S}$ for statistics sampled from the χ^2 distribution and for the Cash statistic (Section 6.3), and is approximately equal to $\sqrt{2\Delta S}$ for fits based on the general log-likelihood ($\log \mathcal{L}$).

Confidence	σ	$\Delta\chi^2$	$\Delta \log \mathcal{L}$
68.3%	1.0	1.00	0.50
90.0%	1.6	2.71	1.36
95.5%	2.0	4.00	2.00
99.0%	2.6	6.63	3.32
99.7%	3.0	9.00	4.50

Table 1.54: Confidence Intervals for the `uncertainty` command

1.85.1 Caveats

An estimated confidence interval is accurate if and only if:

1. no correlations exist between the varied parameter and any of the other nominally thawed parameters,
2. the χ^2 or $\log\mathcal{L}$ surface in parameter space is approximately shaped like a multi-dimensional paraboloid, and
3. the best-fit point is sufficiently far from parameter space boundaries.

One may determine if these conditions hold, for example, by plotting the fit statistic as a function of each parameter's values (the curve should approximate a parabola) and by examining contour plots of the fit statistics made by varying the values of two parameters at a time (the contours should be elliptical, with principal axes aligned along the parameter axes, and parameter space boundaries should be no closer than approximately 3σ from the best-fit point).

If the second and third conditions hold, but the first does not, then the confidence intervals may be determined by using either `COVARIANCE` (Section 1.21) and `PROJECTION` (Section 1.61).

If none of the conditions hold, **then the output from `UNCERTAINTY` is meaningless except that it would give an idea of the scale of the confidence intervals**. To accurately determine the confidence intervals, one would have to reparameterize the model, or use Monte Carlo simulations or Bayesian methods.

Examples:

1. List the current and default values of the `unc` structure, and restore the default values:

```
sherpa> sherpa.unc.sigma = 5
sherpa> list_unc()
Parameter   Current         Default                       Description
----------------------------------------------------------------------
sigma             5               1                   Number of sigma
eps            0.01            0.01               Tolerance for sigma
remin          0.01            0.01   Thresh stat change to reminimize
sherpa> restore_unc()
sherpa> list_unc()
Parameter   Current         Default                       Description
----------------------------------------------------------------------
sigma             1               1                   Number of sigma
eps            0.01            0.01               Tolerance for sigma
remin          0.01            0.01   Thresh stat change to reminimize
```

2. Search parameter space to find a range of parameter values within a confidence interval of 90.0% for all thawed parameters:

```
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> METHOD SIMPLEX
sherpa> FIT
...
sherpa> sherpa.unc.sigma = 1.6
sherpa> UNCERTAINTY

WARNING: found better fit -- reminimizing!
 smplx: v1.3
 smplx:  initial statistic value =     3.33768E-01
 smplx:     converged to minimum =     3.30123E-01 at iteration =     12
```

```
 smplx:    final statistic value =     3.30123E-01
            my.c0  -0.861519
            my.c1  0.672815
            my.c2  0.919114


Computed for uncertainty.sigma = 1.6
        --------------------------------------------------------
        Parameter Name      Best-Fit Lower Bound     Upper Bound
        --------------------------------------------------------
            my.c0         -0.861519  -2.32896        +2.45363
            my.c1          0.672815  -0.757506       +0.730418
            my.c2          0.919114  -0.133193       +0.131418
```

3. Search parameter space to find a range of parameter values within a confidence interval of 68.3% for the parameters my.c0 and my.c1:

```
sherpa> sherpa.unc.sigma = 1.0
sherpa> UNCERTAINTY my.c0 my.c1

Computed for uncertainty.sigma = 1
        --------------------------------------------------------
        Parameter Name      Best-Fit Lower Bound     Upper Bound
        --------------------------------------------------------
            my.c0         -0.861519  -1.4335         +1.56432
            my.c1          0.672815  -0.479026       +0.450944
```

1.85.2 CHANGES IN CIAO 3.2

Prior to CIAO 3.2 the `UNCERTAINTY` command could not be used until the dataset had been fit. This was done to ensure that the parameter values were at their best-fit location, but caused problems when fitting multiple datasets or loading previously-saved analysis sessions. This restriction has now been removed. Please note that the results of `UNCERTAINTY` will not be valid unless the parameters are at their best-fit values.

1.86 UNGROUP

Causes *Sherpa* to undo a grouping scheme that had been applied to source or background data.

```
sherpa> [B]UNGROUP [# [ID]]
```

`UNGROUP` is used to ungroup source data, while `BUNGROUP` is used to ungroup background data.

`#` specifies the number of the dataset to which the grouping scheme is to be applied (default dataset number is 1). The `ID` modifier is used if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The commands `GROUP` and `UNGROUP` allow a user to toggle back and forth between the analysis of grouped and ungrouped data, after grouping assignments have been read into *Sherpa* via the command `READ GROUPS`. (In a future version of *Sherpa*, the `GROUP` may be issued automatically upon the reading in of groups.)

Note the issuing the `UNGROUP` causes *Sherpa* to delete any defined filters for the specified dataset.

NOTE: in CIAO 3.0, the commands `GROUP` and `UNGROUP` may not be used with PHA data that has a `GROUPING` column. This is because these data are grouped before *Sherpa* ever has control of them, and *Sherpa* thus has no knowledge of how the ungrouped data are distributed among bins. This will be changed in a future version of *Sherpa*.

See the documentation on the `GROUP` command for more details and an example.

1.87 UNLINK

Removes a link between model parameters.

```
sherpa> UNLINK <arg>
```

The command-line argument <arg> may be:

Argument	**Description**
<**sherpa_modelname**>.{<**paramname**> | <**#**>}	A specified model component parameter (*e.g.*, `GAUSS.pos`).
<**modelname**>.{<**paramname**> | <**#**>}	A specified model component parameter (*e.g.*, `g.pos`).

Table 1.55: `UNLINK` Command Argument

A link between two model parameters is established using the => operator (see examples, below); `UNLINK` breaks the link. When a link is removed between model parameters, the parameter value will be that of the parameter to which it had been linked. When a link is removed between a model parameter and a model stack, the parameter value will return to its value before such a link was established.

Examples:

1. Set up, and then remove, a link between model parameters:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> GAUSS[modelb]
```

```
modelb.fwhm parameter value [10]
modelb.pos parameter value [0]
modelb.ampl parameter value [1]
sherpa> GAUSS[modelf]
modelf.fwhm parameter value [10]
modelf.pos parameter value [0]
modelf.ampl parameter value [1]
sherpa> modelf.ampl => 0.5*modelb.ampl
```

The last command in this series uses a model parameter expression, to link the `ampl` parameter of `modelf` to 0.5 multiplied by the `ampl` parameter of `modelb`. That is, the amplitudes of two Gaussian models are linked, where one is half that of the other. Note that model parameter expressions cannot be created within the model parameter prompting. The link may be broken as follows:

```
sherpa> UNLINK modelf.ampl
```

2. Set up, and then remove, links between model parameters:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
sherpa> POW[modeld]
sherpa> POW[modele]
sherpa> modelc.1 => modele.1
sherpa> modelc.3 => 2*modeld.3 - modele.3
```

The next-to last command in this series links the first parameter (`gamma`) of `modelc` to the first parameter of `modele`. The last command in this series links the third parameter (`ampl`) of `modelc` to the parameter expression: 2 multiplied by the third parameter of `modeld`, minus the third parameter of `modele`. The links may be broken as follows:

```
sherpa> UNLINK modelc.1
sherpa> UNLINK modelc.3
```

1.88 UNSUBTRACT

Undoes background subtraction.

```
sherpa> UNSUBTRACT [<dataset range> | ALLSETS]
```

where <dataset range> = #, or more generally #:#,#:#,…, such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The `UNSUBTRACT` restores the input source dataset amplitudes to the *Sherpa* session.

See the documentation on the `SUBTRACT` command for more details and an example.

1.89 USE

Calls and executes a *Sherpa* script.

```
sherpa> USE <script_name>
```

where <script_name> is the name of an ASCII file containing Sherpa commands. A *Sherpa* script should contain *Sherpa* commands as they would be typed on the *Sherpa* command-line. Comments may be placed in the script by putting a #, Section 1.5 symbol at the start of the line.

Note that *Sherpa* scripts may also be run directly from the UNIX command line, using the following syntax:

unix% sherpa <script_name>

Note that when a script is run, either from within *Sherpa* or from the UNIX command line, parameter prompting is automatically turned off. Thus there is no need to include a PARAMPROMPT OFF command in the script.

Also note that if a {BYE | EXIT | QUIT} command is not given in the script, control over the *Sherpa* session will revert to the user when *Sherpa* finishes processing the scripted commands.

Sherpa scripts may be created by using the SAVE command, Section 1.72.

Examples:

1. Call and execute a *Sherpa* script:

```
sherpa> $more example.script
# This is my sherpa script
READ DATA data/example1.dat
PARAMPROMPT OFF
POLY[modela]
THAW modela.2 modela.3
SOURCE = modela
FIT
sherpa> USE example.script
```

1.90 VERSION

Reports the Sherpa version that is in use.

```
sherpa> VERSION
Sherpa Version 3.0.2
```

```
sherpa> print( sherpa_version );
30002
```

The version of Sherpa is displayed when it is started up in interactive mode. It can also be found by using the VERSION command, which displays a message on the screen.

The value can also be accessed from a S-Lang script by using the sherpa_version variable. This returns the version using the following formula

$$major * 10000 + minor * 100 + micro \quad (1.7)$$

where the release is given as

$$major.minor.micro \quad (1.8)$$

1.91 WRITE

Causes the specified information to be written to the screen or to a file.

```
sherpa> WRITE <arg> [# [ID]] [<filename> [<filetype>]]
```

`#` specifies the number of the dataset (default dataset number is 1). The `ID` modifier is used for writing background datasets, and then if and only if the *Sherpa* state object variable **`multiback`** is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The name of the file to which data is written is <**`filename`**>. If <**`filename`**> is not specified, information is written to the screen.

The argument <**`arg_n`**> may be any of the following:

Argument	**Displays**
{`DATA` | `DCOUNTS`}	The source data values
{`BACK` | `BDCOUNTS`}	The background data values
`ERRORS`	The errors associated with source data points
`BERRORS`	The errors associated with background data points
{`SYSERRORS` | `BSYSERRORS`}	The assigned systematic errors for the source|background data values
{`STATERRORS` | `BSTATERRORS`}	The estimated statistical errors for the source|background data values
{{`MODEL` | `MCOUNTS`} | {`BMODEL` | `BMCOUNTS`}}	The (convolved) source|background model amplitudes
{`DELCHI` | `BDELCHI`}	The sigma residuals of the source|background fit: (data - model)/error
{`RESIDUALS` | `BRESIDUALS`}	The absolute residuals of the source|background fit: (data - model)
{`RATIO` | `BRATIO`}	The ratio (data/model) for source|background

{CHI SQU | BCHISQU}	The contributions to the χ^2 statistic from each source|background data point
{STATISTIC | BSTATISTIC}	The contributions to the current statistic
{WEIGHT | BWEIGHT}	The statistic weight value assigned to each source|background data point
{FILTER | BFILTER}	The mask value (0|1) for each source|background data point
{GROUP | BGROUP}	The grouping value (1|-1) associated with each source|background data point
{QUALITY | BQUALITY}	The quality value (0|2|5) associated with each source|background data point
<sherpa_model_stack>	The (unconvolved) model amplitudes for the specified model stack (SOURCE, {BACKGROUND|BG}, [B]NOISE, or KERNEL)
<model_stack>	The (unconvolved) model amplitudes for the specified user-defined model stack
<sherpa_modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, GAUSS)
<modelname>	The (unconvolved) amplitudes of the specified model component (*e.g.*, g)
{ARF | BARF}	The unfiltered source|background ARF file contents
{EXPMAP | BEXPMAP}	The unfiltered source|background exposure map file contents
{PSF | BPSF}	The unfiltered source|background PSF file contents
MDL	A model descriptor list file.

Table 1.56: WRITE Command Arguments

The argument <filetype> can specify the desired format for the file that is written:

<filetype>	**Description**
ASCII	1-D ASCII
FITS	2-D FITS image
FITSIMAGE	2-D FITS image
FITSBIN	1-D FITS binary table
PHA	PHA file

Table 1.57: WRITE File Type Arguments

Note the following:

- if <filetype> is not specified, the default output file format is ASCII for 1-D data and FITS for 2-D data.
- the WRITE command utilizes the settings of ANALYSIS and PLOTY to determine the units of the output. An exception to this rule in CIAO 3.0 is the writing out of data to PHA format files; here, counts are

written out regardless of the setting of PLOTY.

- if the data are grouped, then the channels in the file produced by WRITE DATA ... PHA will represent the *grouped* channels. However, since no grouping information is written to the file, the bin energies will be incorrect. Therefore, WRITE ... PHA should be used only with *ungrouped* data.
- the *VARMM*/S-Lang module functions writeascii and writefits, which may be used within *Sherpa*, provide alternate means of writing data to disk.

The WRITE command may be issued at any time. Each time the command is issued the specified file <filename> is overwritten, but only if the state variable sherpa.clobber is set to 1. Otherwise, an error message is displayed if the file exists.

1.91.1 Using Data Model Filters

This command is able to take any Data Model virtual file specification (see "ahelp dmsyntax"). If you can do

```
unix% dmcopy "infile.fits[spec 1][spec 2]" outfile.fits
```

you can also do

```
sherpa> write "infile.fits[spec 1][spec 2]"
```

This is especially useful when working with very large files. For example:

```
sherpa> write "evt.fits[bin sky=4][opt mem=100]"
```

bins the event file by a factor of four and allocates additional memory.

Examples:

1. Write an ASCII dataset to a backup ASCII file:

```
sherpa> READ DATA 2 example.dat 1 2
sherpa> WRITE DATA 2 example_backup.dat
Write X-Axis: Bin  Y-Axis: Flux (Counts)
sherpa> WRITE DATA 2 example_backup.dat
Write X-Axis: Bin  Y-Axis: Flux (Counts)
Error: file exists and sherpa.clobber = 0.
sherpa> sherpa.clobber = 1
sherpa> WRITE DATA 2 example_backup.dat
Write X-Axis: Bin  Y-Axis: Flux (Counts)
```

The second command writes dataset number 2 to an ASCII file named `example_backup.dat`. This file cannot be overwritten unless the state variable `sherpa.clobber` is set to 1.

2. Write a PHA dataset to a backup ASCII file:

```
sherpa> READ DATA example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example.rmf
ARF is being input from:
  <directory_path>/example.arf
Background data are being input from:
  <directory_path>/example_bkg.pha
sherpa> WRITE DATA pha.dat ASCII
Write X-Axis: Energy (keV)  Y-Axis: Flux (Counts/sec/keV)
```

Note that the `ASCII` argument in the command `WRITE DATA pha.dat ASCII` is not actually needed, since the default for 1-D data is ASCII format.

3. Extract three columns from a FITSBIN file and write them to an ASCII file:

```
sherpa> READ DATA "example1_bin.fits[cols x1,x2,x3]" FITSBIN
sherpa> WRITE DATA threecols.txt ASCII 1 2 3
Write X-Axes: (Bin,Bin)  Y-Axis: Counts
```

4. Make a fit to data and save the best-fit amplitudes for one of the model components, along with information about the fit:

```
sherpa> ERASE ALL
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> SOURCE = POLY[p1] + POW[p2]
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 3070.54
 LVMQT: final statistic value = 0.252662 at iteration 14
            p1.c0  0.256496
            p2.gamma  -2.03063
            p2.ampl  15.6881

sherpa> WRITE p2 powlaw.mod ASCII
Write X-Axis: Bin  Y-Axis: Flux (Photons/bin)
sherpa> WRITE SOURCE source.mod
Write X-Axis: Bin  Y-Axis: Flux (Photons/bin)
sherpa> var1 = get_axes(1)
sherpa> print(var1)
axistype        =  Channels
axisunits       =  unknown
```

```
lo              =  NULL
hi              =  NULL
mid             =  Float_Type[8]
sherpa> var2 = get_data(1)
sherpa> var3 = get_errors(1)
sherpa> var4 = get_mcounts(1)
sherpa> writeascii("fit.dat",var1.mid,var2,var3,var4)
```

The last command writes out a 4-column file with x-axis values and associated data, errors, and predicted model counts.

1.92 XSPEC ABUNDAN

Performs the *XSPEC* command **abund**.

```
sherpa> XSPEC ABUNDAN {<arg> | FILE <filename>}
```

where <**arg**> is one of the options listed in the table below.

This commands sets the elemental abundance table used in *XSPEC* plasma-code models. (It does not set abundances for **xswabs**.)

Name	**Description**
angr	default value; Anders E. & Grevesse N. (1989, Geochimica et Cosmochimica Acta 53, 197)
feld	Feldman U. (1992, Physica Scripta 46, 202)
aneb	Anders E. & Ebihara (1982, Geochimica et Cosmochimica Acta 46, 2363)
grsa	Grevesse, N. & Sauval, A.J. (1998, Space Science Reviews 85, 161)
wilm	Wilms, Allen & McCray (2000, ApJ 542, 914)
lodd	The solar photospheric abundances in Lodders, K (2003, ApJ 591, 1220)
file <**filename**>	<filename> is an ASCII file containing 30 lines with one number on each line.

Table 1.58: Allowed Arguments

When using the command, the argument can be shortened as long as it remains unique; see Example 3 below.

All abundances are number relative to H.

Element	angr	feld	aneb	grsa	wilm	lodd
H	1.00e+0	1.00e+0	1.00e+0	1.00e+0	1.00e+0	1.00e+0
He	9.77e-2	9.77e-2	8.01e-2	8.51e-2	9.77e-2	7.92e-2
Li	1.45e-11	1.26e-11	2.19e-9	1.26e-11	0.00	1.90e-9
B	1.41e-11	2.51e-11	2.87e-11	2.51e-11	0.00	2.57e-11
Be	3.98e-10	3.55e-10	8.82e-10	3.55e-10	0.00	6.03e-10
C	3.63e-4	3.98e-4	4.45e-4	3.31e-4	2.40e-4	2.45e-4
N	1.12e-4	1.00e-4	9.12e-5	8.32e-5	7.59e-5	6.76e-5

O	8.51e-4	8.51e-4	7.39e-4	6.76e-4	4.90e-4	4.90e-4
F	3.63e-8	3.63e-8	3.10e-8	3.63e-8	0.00	2.88e-8
Ne	1.23e-4	1.29e-4	1.38e-4	1.20e-4	8.71e-5	7.41e-5
Na	2.14e-6	2.14e-6	2.10e-6	2.14e-6	1.45e-6	1.99e-6
Mg	3.80e-5	3.80e-5	3.95e-5	3.80e-5	2.51e-5	3.55e-5
Al	2.95e-6	2.95e-6	3.12e-6	2.95e-6	2.14e-6	2.88e-6
Si	3.55e-5	3.55e-5	3.68e-5	3.55e-5	1.86e-5	3.47e-5
P	2.82e-7	2.82e-7	3.82e-7	2.82e-7	2.63e-7	2.88e-7
S	1.62e-5	1.62e-5	1.89e-5	2.14e-5	1.23e-5	1.55e-5
Cl	1.88e-7	1.88e-7	1.93e-7	3.16e-7	1.32e-7	1.82e-7
Ar	3.63e-6	4.47e-6	3.82e-6	2.51e-6	2.57e-6	3.55e-6
K	1.32e-7	1.32e-7	1.39e-7	1.32e-7	0.00	1.29e-7
Ca	2.29e-6	2.29e-6	2.25e-6	2.29e-6	1.58e-6	2.19e-6
Sc	1.26e-9	1.48e-9	1.24e-9	1.48e-9	0.00	1.17e-9
Ti	9.77e-8	1.05e-7	8.82e-8	1.05e-7	6.46e-8	8.32e-8
V	1.00e-8	1.00e-8	1.08e-8	1.00e-8	0.00	1.00e-8
Cr	4.68e-7	4.84e-7	4.93e-7	4.68e-7	3.24e-7	4.47e-7
Mn	2.45e-7	2.45e-7	3.50e-7	2.45e-7	2.19e-7	3.16e-7
Fe	4.68e-5	3.24e-5	3.31e-5	3.16e-5	2.69e-5	2.95e-5
Co	8.60e-8	8.60e-8	8.27e-8	8.32e-8	8.32e-8	8.13e-8
Ni	1.78e-6	1.78e-6	1.81e-6	1.78e-6	1.12e-6	1.66e-6
Cu	1.62e-8	1.62e-8	1.89e-8	1.62e-8	0.00	1.82e-8
Zn	3.98e-8	3.98e-8	4.63e-8	3.98e-8	0.00	4.27e-8

Table 1.59: Tables for xspecabundan Arguments

This information is taken from the XSpec User's Guide. Version 11.3.1.x of the *XSpec* models is supplied with CIAO 3.4.

Examples:

1. Run the *XSPEC* command `abund`, setting abundances to Feldman:

```
sherpa> XSPEC ABUNDAN FELD
    Abundances set to Feldman
```

2. Reset the abundances to the default value:

```
sherpa> XSPEC ABUNDAN ANGR
    Abundances set to Anders & Grevesse
```

3. The argument can be shortened as long as it remains unique.

```
sherpa> XSPEC ABUNDAN W
    Abundances set to Wilms et al.
sherpa> XSPEC ABUNDAN F
    Abundances set to Feldman
sherpa> XSPEC ABUNDAN G
    Abundances set to Grevesse & Sauval
```

In the case of the two arguments that begin with "AN", "ANGR" takes precedence over "ANEB" if only the first or second letter is given:

```
sherpa> XSPEC ABUNDAN A
    Abundances set to Anders & Grevesse
sherpa> XSPEC ABUNDAN AN
    Abundances set to Anders & Grevesse
sherpa> XSPEC ABUNDAN ANE
    Abundances set to Anders & Ebihara
```

4. Read custom abundances from a file:

```
sherpa> XSPEC ABUNDAN FILE abund.txt
    Abundances set to those read from file
```

1.93 XSPEC XSECT

Performs the *XSPEC* command `xsect`.

```
sherpa> XSPEC XSECT <arg>
```

where <`arg`> is one of the options listed in the table below.

This commands sets the photoelectric cross-section used in *XSPEC* absorption models. (It does not set the cross-section for `xswabs`.)

Name	Description
`bcmc`	default value; Balucinska-Church & McCammon (1992, ApJ 400, 699) with a new He cross-section based on (1998, ApJ 496, 1044)
`obcm`	bcmc with the old He cross-section
`vern`	Verner *et al.* (1996, ApJ 465, 487)

Table 1.60: Allowed Arguments

Examples:

1. Run the *XSPEC* command `xsect`, setting abundances to `OBCM`:

```
sherpa> XSPEC XSECT OBCM
```

2. Reset the cross-section to the default value:

```
sherpa> XSPEC XSECT BCMC
```

Chapter 2

Sherpa Display

2.1 Introduction

In this Chapter, the commands that are available from within *Sherpa* for displaying 1- and 2-D data are described.

2.2 1-D Plotting

1-D plotting within *Sherpa* may be performed using the `LPLOT` command:

sherpa> `LPLOT` <**arg**> [#] [{UP | DOWN}]

where # may specify the dataset (default dataset number is 1).

<**arg**> may be:

Argument:	**To display:**
BACK	The background data values.
BACKERRORS	The background error values.
{BACKFIT | BFIT}	The background data and the best-fit background model simultaneously.
{BACKGROUND | BG}	The (unconvolved) background model amplitudes.
BCHISQU	The contributions to the χ^2 statistic in the background fit.
BDELCHI	The residuals of the background fit ($(data - model)/error$).
BFILTER	The filter status of each background data point.
{BFIT | BACKFIT}	The background data and the best-fit background model simultaneously.
{BG | BACKGROUND}	The (unconvolved) background model amplitudes.
BRATIO	The background data/model ratio value for each data point.
BRESIDUALS	The residuals of the background fit ($data - model$).
BSTATISTIC	The contributions to the currently defined background fit statistic.
CHI SQU	The contributions to the χ^2 statistic.
DATA	The source data.

Argument:	**To display:**
DELCHI	The residuals of the fit ($(data - model)/error$).
ERRORS	The source data error values.
FILTER	The filter status of each data point.
FIT	The source data and the best-fit source model simultaneously.
<modelname>	The specified model component amplitudes (unconvolved).
MODEL	The (convolved) source model amplitudes.
RATIO	The data/model ratio value for each data point.
RESIDUALS	The residuals of the fit ($data - model$).
SOURCE	The (unconvolved) source model amplitudes.
STATISTIC	The contributions to the currently defined fit statistic.
USERSTAT	The contributions to the user-defined fit statistic.
WEIGHTS	The weight assignments for each data point.

The modifiers UP and DOWN may be used with BACK, BACKERRORS, BACKFIT, and BACKGROUND. They relate specifically to the analysis of *Chandra* grating data and thus may be ignored in most applications.

Note: If there is no open plotting window when an LPLOT command is given, one will be launched automatically. Plotting display windows may be closed simply by closing the window with a mouse click.

In addition, 1-D plotting within *Sherpa* , of instrument responses stored in a Fits Embedded Function (FEF) file that is read in via FEFFILE (See Chapter 1, Section 1.33), may be plotted using FEFPLOT:

```
sherpa> FEFPLOT {<photon_energy> | <photon_wavelength>}
```

where {<photon_energy> | <photon_wavelength>} must be in the same units that are used for photon energies in the FEF file (conventionally, keV or Å).

Examples:

1. Display 1-D data:

```
sherpa> READ DATA 2 example.dat
sherpa> LPLOT DATA 2
```

   The last command plots dataset number 2. Dataset number 2 must be a 1-D dataset.

2. Display a fit to 1-D data:

```
sherpa> PARAMPROMPT OFF
sherpa> SOURCE 2 = POLY[modela]
sherpa> THAW modela.1 modela.2 modela.3
sherpa> GUESS SOURCE 2
sherpa> FIT 2
sherpa> LPLOT FIT 2
```

   Following a fit to dataset number 2, the last command plots the fit.

3. Read in part of a FEF file, and plot the response as a function of counts-space energy, at photon energy 2.4 keV:

```
sherpa> FEFFILE "data/fef_response.fits[function] \
[ccd_id=0,chipx>=1,chipx<=256,chipy>=1,chipy<=32]"
sherpa> FEFPLOT 2.4
```

The appearance of the plot may be subsequently altered using *ChIPS* commands such as:

```
sherpa> LIMITS X 0 3
sherpa> REDRAW
```

Additional `LPLOT` command examples may be found throughout this Chapter, and in Chapter 1, Section 1.51.

2.2.1 1-D Plot Controls

There are two types of control commands for 1-D plots available from within *Sherpa*:

1. *Sherpa* 1-D plot control commands. This type of plot control command requires that an `LPLOT` command be issued, in order for the plot display to be updated. That is, the plot display is *not* automatically redrawn when a *Sherpa* plot control command is issued. See Subsection 2.2.2, below.
2. *ChIPS* 1-D plot control commands. This type of plot control command requires that a `REDRAW` command be issued, in order for the plot display to be updated. That is, the plot display is *not* automatically redrawn when a *ChIPS* plot control command is issued. See Subsection 2.2.3, below.

2.2.2 *Sherpa* 1-D Plot Controls

- The *Sherpa* `SET` command may be used to control a number of plotting characteristics:

  `sherpa> SET <lplot_option>`

  where `<lplot_option>` may be:

Argument:	**For the plot to:**
`PLOT AUTORANGE`	Set the x-axis (Axis 0) range automatically, based on the data.
`PLOT ERRORBARS`	Include symmetrical x-axis and y-axis error bars.
	Plots the data using a histogram.
`PLOT LINEAR LINEAR`	Use a linear scale along both axes.
`PLOT LINEAR LOG`	Use a linear scale along the x-axis, and a logarithmic scale along the y-axis.
	Connects the data points with a line.
`PLOT LOG LINEAR`	Use a logarithmic scale along the x-axis, and a linear scale along the y-axis.
`PLOT LOG LOG`	Use a logarithmic scale along both axes.
`PLOT NOERRORBARS`	Not include error bars.
`PLOT RANGE <min>:<max>`	Set the x-axis range for the plot.
`PLOT XERRORBARS`	Include x-axis error bars only.
`PLOT YERRORBARS`	Include y-axis error bars only (default setting).

Table 2.2: Valid `SET` Command Arguments

- The *Sherpa* PLOTX command may be used to set the unit type for the x-axis (Axis 0) of the plot:

  sherpa> PLOTX [#] {BIN | CHANNEL | ENERGY | WAVE}

  where # may specify the number of the dataset (the default dataset is 1). The arguments {BIN | CHANNEL} specify that the x-axis units are to be the sequential bin numbers of the input data. For ENERGY, the units are those used to define the energies in the EBOUNDS extension of the input RMF file (although the plot *label* will specify keV by default). For WAVE, the units are those used to define the wavelengths in the BIN_LO and BIN_HI columns of the input PHA data (although the plot *label* will specify Åby default). If necessary, the user can change the label with the *ChIPS* command XLABEL.

  Defining an instrument model automatically causes PLOTX to be changed to ENERGY or WAVE; the user does not normally have to apply this command. Also, changing PLOTX to ENERGY or WAVE will have no effect if an instrument model has not been defined (see Chapter 1, Section 1.46).

- The *Sherpa* PLOTY command may be used to set the unit type for the y-axis (Axis 1) of the plot:

  sherpa> PLOTY [#] {COUNTS | RATE}

  where # may specify the number of the dataset (the default dataset is 1). If the user specifies COUNTS, the raw input data will be displayed (even if these data are not actually counts, but something else, such as fluxes). What is displayed if the argument RATE is given depends on whether (a) the exposure time is provided; (b) an ancillary response file (ARF) is provided; (c) the energy or wavelength grid is provided; and (d) the instrument model is defined (see Chapter 1, Section 1.46). The plot labeling function assumes exposure times in seconds, ARFs with effective area information in cm^2, and energies/wavelengths in keV/Å. If necessary, the user can change the label with the *ChIPS* command YLABEL.

- The *Sherpa* LPLOT command may be used to plot into multiple windows:

  sherpa> LPLOT [#] <wind1> <wind2> ...

  where # sets the number of plotting windows and the arguments <wind1> ... should each be replaced with a valid plotting command argument, which will be applied to the respective window. The arguments <wind1> ... must be of the form <arg> {# | #:# | ALL}, where <arg> is any of the LPLOT arguments listed above, and where # specifies a dataset number (default dataset number is 1), or #:# represents an inclusive range of datasets.

  Note that in the current version of *Sherpa*, only *ChIPS* plot control commands should be applied when working with multiple display areas (*i.e.* the *Sherpa* plot control commands SET, PLOTX, and PLOTY currently do not work properly with multiple plotting windows). See Subsection 2.2.3, Example 3.

Note that the LPLOT command must be issued, in order to see the results of previously issued SET, PLOTX, and/or PLOTY commands.

Examples:

1. Plot data and change error bars, x-axis range, and scale:

```
sherpa> DATA data/example.dat 1 2
sherpa> LPLOT DATA
sherpa> SET PLOT NOERRORBARS
sherpa> LPLOT DATA
sherpa> SET PLOT RANGE 1600:1640
sherpa> LPLOT DATA
sherpa> SET PLOT LOG LINEAR
sherpa> LPLOT DATA
```

The first `LPLOT DATA` command produces a plot with default settings. The command `SET PLOT NOERRORBARS` sets for no error bars to be plotted, and then the `LPLOT DATA` command causes the plot to be redrawn with this new setting. Then the command `SET PLOT RANGE 1600:1640` changes the Axis 0 (x-axis) range; the `LPLOT DATA` command causes the plot to be redrawn with this new setting. Next, the Axis 0 scale is changed to log scale, and again the plot is redrawn using the `LPLOT DATA` command.

Note that the following sequence of commands would also produce the same plot:

```
sherpa> DATA data/example.dat 1 2
sherpa> SET PLOT NOERRORBARS
sherpa> SET PLOT RANGE 1600:1640
sherpa> SET PLOT LOG LINEAR
sherpa> LPLOT DATA
```

Also note that the same plot may be produced using *ChIPS* plot control commands; see Section 2.2.3, Example 1.

2. Restore plot to default errorbars, x-axis range, and scale:

```
sherpa> SET PLOT ERRORBARS
sherpa> SET PLOT AUTORANGE
sherpa> SET PLOT LINEAR LINEAR
sherpa> LPLOT DATA
```

Following the above example, this set of commands restores the plot to the original default setting.

The same plot may be produced using *ChIPS* plot control commands; see Section 2.2.3, Example 2.

3. Change the x-axis and y-axis unit types, for the display of dataset number 2:

```
sherpa> READ DATA 2 data/example.pha
sherpa> RSP[instrumentA](data/example.rmf, data/example.arf)
sherpa> INSTRUMENT 2 = instrumentA
sherpa> SET PLOT NOERRORBARS
sherpa> PLOTX 2 ENERGY
sherpa> LPLOT DATA 2
sherpa> PLOTX 2 CHANNEL
sherpa> LPLOT DATA 2
sherpa> PLOTY 2 COUNTS
sherpa> LPLOT DATA 2
```

The `PLOTX 2 ENERGY` command sets the unit type, for the x-axis of the plot, for dataset number 2, to energy. The next `LPLOT` command then plots dataset number 2 using this new Axis 0 setting. Note that the `PLOTX` command cannot be successfully issued without first defining the instrument model to be used (see Chapter 1, Section 1.46). The `PLOTY 2 COUNTS` command sets the unit type, for the y-axis of the plot, for dataset number 2, to counts. The next `LPLOT` command then plots dataset number 2 using this new Axis 1 setting.

4. Plot into multiple windows:

```
sherpa> PLOTX 2 ENERGY
sherpa> PLOTY 2 RATE
sherpa> LPLOT DATA 2
sherpa> LPLOT 2 DATA 1 DATA 2
```

The final command creates two drawing areas, and plots dataset number 1 in the first drawing area, and dataset number 2 in the second drawing area. Note that in this example, the command `LPLOT 2 DATA 1 DATA 2` could be replaced with the simpler command `LPLOT 2 DATA 1:2`, or `LPLOT 2 DATA ALL`.

5. Have plotted data written to output data files:

```
sherpa> ERASE ALL
sherpa> DATA data/example1a.dat
sherpa> SET PLOT NOERRORBARS
sherpa> LPLOT DATA
sherpa> STORE myplot1.chp
sherpa> SAVE ALL mysession1.shp
sherpa> ls myplot1.chp*
myplot1.chp
myplot1.chp.fits
sherpa> ls mysession1.shp*
mysession1.shp
sherpa> EXIT
Goodbye.
```

The `STORE` command specifies that a record of the commands used to generate the plot be saved in an ASCII file named `myplot1.chp`. The `STORE` command also creates a FITS file that contains the plotted data (*e.g.* `myplot1.chp.fits`). The command `SAVE ALL mysession1.shp` is useful for saving the current state of the *Sherpa* session; the current session may be restored at a later time by using the `mysession1.shp` file as a *Sherpa* script. For example, after exiting *Sherpa* , one may restore the stored plot using the *ChIPS* command `RESTORE`:

```
sherpa> RESTORE myplot1.chp
sherpa> SHOW
```

And, one may also resume the saved *Sherpa* session using the *Sherpa* command `USE`:

```
sherpa> USE mysession1.shp
sherpa> SHOW
```

Note that when multiple curves have been plotted, the data for each curve is saved in a different extension of the `STORE` command's output FITS file. For example:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GAUSS[modela]
sherpa> SOURCE 1 = modela
sherpa> FIT
 powll: v1.2
 powll:   initial function value =     7.75333E-28
 powll:     converged to minimum =     3.54037E-28 at iteration =      9
 powll:   final function value    =     3.54037E-28
       modela.fwhm  15.8599
       modela.pos  1615.64
       modela.ampl  1.11265e-14
```

```
sherpa> SET PLOT NOERRORBARS
sherpa> LPLOT FIT
sherpa> STORE myplot2.chp
sherpa> ls myplot2.chp*
myplot2.chp
myplot2.chp.fits
sherpa> PRISM myplot2.chp.fits
```

Examination of the FITS file `myplot2.chp.fits` using *Prism*should show that the data are saved in the first FITS extension, and the fit data are saved in the second extension. See the *ChIPS* Reference Manual for further information regarding the *ChIPS* `STORE` and `RESTORE` commands.

Additional `SET` command examples may be found in Chapter 1, Section 8.80. Additional `PLOTX` and `PLOTY` examples may be found in Chapter 1, Sections 1.59 and 1.60, respectively.

2.2.3 *ChIPS* 1-D Plot Controls

All *ChIPS* commands are available from within *Sherpa*. See the *ChIPS* Reference Manual for complete *ChIPS* command definitions and further examples.

Note that *ChIPS* plot control commands require that a `REDRAW` command be issued, in order for the plot display to be updated.

Examples:

1. Plot data and change x-axis range and scale:

```
sherpa> DATA data/example.dat 1 2
sherpa> LPLOT DATA
sherpa> ERRS NONE
sherpa> LIMITS X 1600 1640
sherpa> AXIS LOG X
sherpa> REDRAW
```

The first `LPLOT DATA` command produces a plot with default settings. The *ChIPS* command `ERRS NONE` sets for no error bars to be plotted. The *ChIPS* command `LIMITS X 1600 1640` changes the Axis 0 (x-axis) range. Next, the Axis 0 scale is changed to log scale. Note that the plot display is *not* automatically redrawn when *ChIPS* plot control commands are issued; the *ChIPS* command `REDRAW` must be issued.

The same plot may be produced using *Sherpa* plot control commands; see Section 2.2.2, Example 1.

2. Restore plot to default x-axis range and scale:

```
sherpa> ERRS STANDARD
sherpa> AXIS LINEAR X
sherpa> LIMITS X AUTO AUTO
sherpa> REDRAW
```

Following the above example, this set of commands restores the plot to the original default setting. The same plot may be produced using *Sherpa* plot control commands; see Section 2.2.2, Example 2.

3. Plot two data datasets, each in separate windows; change x-axis range and scale of one plot:

```
sherpa> ERASE ALL
sherpa> READ DATA 1 data/example.dat 1 2
sherpa> READ DATA 2 data/example2.dat 1 2
sherpa> LPLOT 2 DATA 1 DATA 2
sherpa> D 1 LABEL 1.0 15.0 "Drawing Area Number 1"
sherpa> D 2 LABEL 1.0 55.0 "Drawing Area Number 2"
sherpa> D 1 LIMITS X 1600 1640
sherpa> D 1 LOG X
sherpa> REDRAW
```

The *Sherpa* plot control command `LPLOT 2 DATA 1 DATA 2` produces two plot windows, each with default settings. Dataset number 1 is plotted in drawing area number 1, and dataset number 2 is plotted in drawing area number 2. The *ChIPS* plot control command `D 1 LIMITS X 1600 1640` changes the Axis 0 (x-axis) range for the plot in the first drawing area. Next, another *ChIPS* plot control command changes the Axis 0 scale to log scale, for the plot in drawing area number 1.

4. Utilize various *ChIPS* plot control commands:

```
sherpa> ERASE ALL
sherpa> DATA data/example.dat 1 2
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POLY[model1]
sherpa> SOURCE = model1
sherpa> THAW model1.c0 model1.c1 model1.c2
sherpa> FIT
sherpa> LPLOT FIT
```

With the above commands, an ASCII dataset is input and then fit with a polynomial. The `LPLOT FIT` command produces a plot of the data and the resulting fit, with default settings.

```
sherpa> C 1 NOLINE
sherpa> C 1 SYMBOL CROSS
sherpa> C 2 SIMPLELINE
sherpa> REDRAW
```

This plot possesses two curves: curve 1 is the data curve, and curve 2 is the fit function. In this example, *ChIPS* plot control commands are issued, to modify each curve of the plot. First, the data curve is changed to points, and then the points are changed to the cross symbol. Lastly, the fit function curve is changed to a line.

```
sherpa> LIMITS Y 0 4.25
sherpa> TICKVALS Y %1.1f
sherpa> REDRAW
```

The Axis 1 (y-axis) lower and upper limits are modified, using the *ChIPS* `LIMITS` command. Then the *ChIPS* command `TICKVALS` is used to set the format for the y-axis tick labels.

```
sherpa> XLABEL "Arc Minutes"
sherpa> YLABEL "SNR"
sherpa> TITLE "HRC-I"
sherpa> LABEL 5 0.5 "(0.606)+(0.419)X-(0.014)X^2"
sherpa> REDRAW
```

Various *ChIPS* commands are issued to add Axis labels and a title. The last command places a label at the position of (5, 0.5).

```
sherpa> XLABEL BLUE
sherpa> YLABEL BLUE
sherpa> TITLE BLUE
sherpa> LABEL RED
sherpa> REDRAW
```

The label and title color attributes are modified using *ChIPS* commands.

5. Save a plot as a PostScript file:

```
sherpa> PRINT POSTFILE example_plot.ps
```

This command prints the current plot to a PostScript file named `example_plot.ps`.

6. Plot data with the *ChIPS* `CURVE` command:

```
sherpa> ERASE ALL
sherpa> CURVE data/example.dat x 1 y 2
sherpa> REDRAW
```

Within *Sherpa*, a dataset may be directly plotted from a file, using the *ChIPS* command `CURVE`. The above `CURVE` command plots the file `data/example.dat`; the first two columns of this ASCII data file are the x-axis and y-axis data.

```
sherpa> C 1 DEL
sherpa> CURVE data/example.dat x 3 y 4
sherpa> REDRAW
```

This `CURVE` command plots the third and fourth columns as the x and y data. Note that the first curve is deleted before another `CURVE` command is issued. This deletion is necessary in this example, because each `CURVE` commands appends a curve to the plot (that is, successive `CURVE` commands *do not overwrite* each other, they instead overlay each other).

7. Plot data with x-axis error bars:

```
sherpa> ERASE ALL
sherpa> CURVE data/example.dat x 1 y 2 XUP 3 XDOWN 4
sherpa> REDRAW
```

In order to utilize *ChIPS* commands for plotting errorbars, the *ChIPS* `CURVE` command must be used. The `CURVE` command given above plots data from the file `data/example.dat`; the first two columns are the x and y data, while the third and fourth columns are the positive and negative x-axis errorbars, respectively.

```
sherpa> C 1 DEL
sherpa> CURVE data/example.dat x 1 y 2 xerr 3
sherpa> REDRAW
```

This CURVE command plots the first two columns as the x and y data, while the third column is read for symmetrical x-axis errorbars. Note that the first curve is deleted before another CURVE command for the same data is issued.

8. Plot data with y-axis error bars:

```
sherpa> ERASE ALL
sherpa> CURVE data/example.dat x 1 y 2 YUP 5 YDOWN 6
sherpa> REDRAW
```

The CURVE command given above plots data from the file data/example.dat; the first two columns are the x and y data, while the fifth and sixth columns are the positive and negative y-axis errorbars, respectively. The plot may be changed to include only the positive y-axis errorbars, only the negative y-axis errorbars, both, or none:

```
sherpa> ERRS y UP
sherpa> REDRAW
sherpa> ERRS y DOWN
sherpa> REDRAW
sherpa> ERRS y BOTH
sherpa> REDRAW
sherpa> ERRS y NONE
sherpa> REDRAW
```

9. Plot data with symmetrical y-axis error bars:

```
sherpa> ERASE ALL
sherpa> CURVE data/example5b.dat x 1 y 2 yerr 5
sherpa> REDRAW
```

This command plots the first two columns of data, and uses column five to plot symmetrical y-axis errorbars. The following commands are each equivalent to the above CURVE command:

```
sherpa> CURVE data/example5b.dat x 1 y 2 e 5
sherpa> CURVE data/example5b.dat 1 2 5
```

2.2.4 Interactive Filtering from 1-D Plots

Filter regions may be specified from a plot, using the following steps:

1. Plot the data using LPLOT.
2. Issue the command IGNORE [<dataset range> | ALLSETS] PLOT or the command NOTICE [<dataset range> | ALLSETS] PLOT.
3. Place the cursor on the plot, at the desired minimum x-axis value for the filter, and click once with the left mouse button.

4. Place the cursor at the desired maximum x-axis value, and click again with the left mouse button.

This will set the data filter to either include (for `NOTICE`) or exclude (for `IGNORE`) the marked region of the dataset. Note that the character `e` may be used as an alternative to clicking the left mouse button.

See Chapter 1, Section 1.44, Examples 1 and 4.

2.2.5 Interactive Assignment of Parameter Values from 1-D Plots

Model parameter values may be assigned from a plot, using the following steps:

- To take an x-axis value from a plot, and assign it to a model parameter:
  1. Plot the data (see the `LPLOT` command, Chapter 1, Section 1.51).
  2. Issue the command `GETX` <`modelname`>`.`{<`paramname`> `|` <`#`>}.
  3. Place the cursor on the plot, at the desired x-axis value for the parameter, and click once with the left mouse button. Alternatively, hit the character `e`, at the desired x-axis value for the parameter.
- To take two x-axis values from a plot, and assign the non-negative value of their difference to a model parameter:
  1. Plot the data (see the `LPLOT` command, Chapter 1, Section 1.51).
  2. Issue the command `GETX 2` <`modelname`>`.`{<`paramname`> `|` <`#`>}.
  3. Place the cursor on the plot, at the first desired x-axis value, and click once with the left mouse button. Place the cursor at the next desired x-axis value, and left-mouse-click again. Note that the character `e` may be used as an alternative to clicking the left mouse button.
- To take a y-axis value from a plot, and assign it to a model parameter:
  1. Plot the data (see the `LPLOT` command, Chapter 1, Section 1.51).
  2. Issue the command `GETY` <`modelname`>`.`{<`paramname`> `|` <`#`>}.
  3. Place the cursor on the plot, at the desired y-axis value for the parameter, and click once with the left mouse button. Alternatively, hit the character `e`, at the desired y-axis value for the parameter.
- To take two y-axis values from a plot, and assign the non-negative value of their difference to a model parameter:
  1. Plot the data (see the `LPLOT` command, Chapter 1, Section 1.51).
  2. Issue the command `GETY 2` <`modelname`>`.`{<`paramname`> `|` <`#`>}.
  3. Place the cursor on the plot, at the first desired y-axis value, and click once with the left mouse button. Place the cursor at the next desired y-axis value, and left-mouse-click again. Note that the character `e` may be used as an alternative to clicking the left mouse button.

One may quit `GETX` or `GETY` by typing `q`, with the the previously assigned parameter value restored. Note, however, that if multiple parameter values are being set, only the one being altered when `q` is typed has its value restored.

See Chapter 1, Sections 1.39 and 1.40 for examples.

2.2.6 1-D Overplotting

1-D overplotting within *Sherpa* may be performed using the `OPLOT` command:

```
sherpa> OPLOT <curve_arg1> <curve_arg2> ...
```

where the arguments `<curve_arg1> <curve_arg2> ...` should each be replaced with a valid plotting command argument of the form `<arg> [#] [{UP | DOWN}]`, such that `<arg>` is any of the `OPLOT` arguments listed below, and where # specifies a dataset number (default dataset number is 1).

`<arg>` may be:

Argument:	**To display:**
BACK	The background data values.
BACKERRORS	The background error values.
BFILTER	The filter status of each background data point.
BACKGROUND	The (unconvolved) background model amplitudes.
BRATIO	The background data/model ratio value for each data point.
BRESIDUALS	The residuals of the background fit, in units of sigma.
BSTATISTIC	The contributions to the currently defined background fit statistic.
CHI SQU	The contributions to the χ^2 statistic.
DATA	The source data.
ERRORS	The source data error values.
FILTER	The filter status of each data point.
<modelname>	The specified model component amplitudes (unconvolved).
MODEL	The (convolved) source model amplitudes.
RATIO	The data/model ratio value for each data point.
RESIDUALS	The residuals of the fit, in units of sigma.
SOURCE	The (unconvolved) source model amplitudes.
STATISTIC	The contributions to the currently defined fit statistic.
USERSTAT	The contributions to the user-defined fit statistic.
WEIGHTS	The weight assignments for each data point.

The modifiers `UP` and `DOWN` may be used with `BACK`, `BACKERRORS`, `BACKFIT`, and `BACKGROUND`; they relate specifically to the analysis of *Chandra* grating data and thus may be ignored in most applications.

Note: If there is no open plotting window when an `OPLOT` command is given, one will be launched automatically. Plotting display windows, may be closed simply by closing the window with a mouse click.

Examples:

1. Display 1-D data:

   ```
   sherpa> DATA data/example1.dat
   sherpa> DATA 2 data/example2.dat
   sherpa> OPLOT DATA 1 DATA 2
   ```

   The `OPLOT` command causes both dataset number 1 and dataset number 2 to be plotted in the same window.

2.3 2-D Imaging

2-D imaging within *Sherpa* may be performed using the `IMAGE` command:

`sherpa> IMAGE <arg> [#] [{FILTERED | UNFILTERED}]`

where `#` may specify the dataset (default dataset number is 1). By default, filtered datasets are imaged.

`<arg>` may be:

Argument:	**To display:**
BACK	The background data values.
BACKERRORS	The background error values.
{BACKFIT | BFIT}	The background data and the best-fit background model simultaneously.
{BACKGROUND | BG}	The (unconvolved) background model amplitudes.
BCHISQU	The contributions to the χ^2 statistic in the background fit.
BDELCHI	The residuals of the background fit ($(data - model)/error$).
BFILTER	The filter status of each background data point.
{BFIT | BACKFIT}	The background data and the best-fit background model simultaneously.
{BG | BACKGROUND}	The (unconvolved) background model amplitudes.
BRATIO	The background data/model ratio value for each data point.
BRESIDUALS	The residuals of the background fit ($data - model$).
BSTATISTIC	The contributions to the currently defined background fit statistic.
CHI SQU	The contributions to the χ^2 statistic.
DATA	The source data.
DELCHI	The residuals of the fit ($(data - model)/error$).
ERRORS	The data error values.
FILTER	The filter status of each data point.
FIT	The data, the best-fit model, and the sigma residuals.
<modelname>	The specified model component amplitudes (unconvolved).
MODEL	The (convolved) source model amplitudes.
RATIO	The data/model ratio value for each data point.
RESIDUALS	The residuals of the fit ($data - model$).
SOURCE	The (unconvolved) source model amplitudes.
STATISTIC	The contributions to the currently defined fit statistic.
USERSTAT	The contributions to the user-defined fit statistic.
WEIGHTS	The weight assignment for each data point.

Note that if there is no open imager when an `IMAGE` command is given, one will be created automatically. Image display windows may be closed simply by closing the window with a mouse click. Also, the command `CLOSE` may be used to close an image display window.

Examples:

1. Display 2-D data:

```
sherpa> READ DATA 2 example_img.fit
sherpa> IMAGE DATA 2
```

   The last command displays dataset number 2. Dataset number 2 must be a 2-D dataset.

Additional IMAGE command examples may be found in Chapter 1, Section 1.45.

2.3.1 Interactive Filtering from 2-D Images

Filter regions may be specified from an image, using the following steps:

1. Display the image and mark the filter region(s) (see the IMAGE command, Chapter 1, Section 1.45). **Note:** use *include* regions (as opposed to *exclude* regions) for optimal results. For instance, do not do IGNORE IMAGE using an *exclude* region; it will lead to an incorrect result in the current version of *Sherpa*. Use NOTICE IMAGE with an *include* region instead. Placing *exclude* region boundaries within *include* region boundaries will usually work, however. *Verify results using* IMAGE FILTER.
2. Issue the command IGNORE [<**dataset range**> | ALLSETS] IMAGE or the command NOTICE [<**dataset range**> | ALLSETS] IMAGE.

This will set the data filter to either include (for NOTICE) or exclude (for IGNORE) the marked region of the dataset.

Also note: If you re-display an image following filtering, the filtered image data will be displayed if possible. There are some situations when this is not possible, however: currently, arbitrary filters cannot be passed from *Sherpa* to the imaging display. Instead, only rectangles can be passed. And, the rectangle that's passed is one that's the smallest possible around the noticed data. If this rectangle is not smaller than the dataset, then the entire dataset will be shown.

See Chapter 1, Section 1.44, Example 3; and Chapter 1, Section 1.55, Example 4.

2.4 2-D Plotting

2-D plotting within *Sherpa* may be performed using the CPLOT and SPLOT commands:

The CPLOT command causes the specified 2-D data to be displayed, with a contour plot, via *ChIPS* (*Chandra* Imaging and Plotting System):

sherpa> CPLOT <**arg**> [#]

The SPLOT command causes the specified 2-D data to be displayed, with a surface plot, via *ChIPS* (*Chandra* Imaging and Plotting System):

sherpa> SPLOT <**arg**> [#]

where # may specify the dataset (default dataset number is 1).

<**arg**> for either CPLOT or SPLOT may be:

Argument:	**To display:**
BACK	The background data values.

Argument:	To display:
BACKERRORS	The background error values.
{BACKGROUND | BG}	The (unconvolved) background model amplitudes.
BCHISQU	The contributions to the χ^2 statistic in the background fit.
BDELCHI	The residuals of the background fit $((data - model)/error)$.
BFILTER	The filter status of each background data point.
{BG | BACKGROUND}	The (unconvolved) background model amplitudes.
BRATIO	The background data/model ratio value for each data point.
BRESIDUALS	The residuals of the background fit $(data - model)$.
BSTATISTIC	The contributions to the currently defined background fit statistic.
CHI SQU	The contributions to the χ^2 statistic.
DATA	The source data.
DELCHI	The residuals of the fit $((data - model)/error)$.
ERRORS	The source data error values.
FILTER	The filter status of each data point.
<modelname>	The specified model component amplitudes (unconvolved).
MODEL	The (convolved) source model amplitudes.
RATIO	The data/model ratio value for each data point.
RESIDUALS	The residuals of the fit $(data - model)$.
SOURCE	The (unconvolved) source model amplitudes.
STATISTIC	The contributions to the currently defined fit statistic.
USERSTAT	The contributions to the user-defined fit statistic.
WEIGHTS	The weight assignments for each data point.

Note that, if there is not a plotting window currently open, any CPLOT or SPLOT command will automatically launch one. Plotting display windows may be closed simply by closing the window with a mouse click.

In addition, like LPLOT, surface and contour plots may be displayed into multiple windows:

```
sherpa> CPLOT [#] <wind1> <wind2> ...
sherpa> SPLOT [#] <wind1> <wind2> ...
```

where # sets the number of plotting windows and the arguments <wind1> ... should each be replaced with a valid surface plotting command argument, which will be applied to the respective window. The arguments <wind1> ... must be of the form <arg> {# | #:# | ALL}, where <arg> is any of the SPLOT arguments listed above, and where # specifies a dataset number (default dataset number is 1), or #:# represents an inclusive range of datasets.

In addition, 2-D plotting within *Sherpa* may be performed using the REGION-PROJECTION and REGION-UNCERTAINTY commands:

REGION-PROJECTION creates a contour plot of confidence regions, computed using the PROJECTION algorithm (See Chapter 1, Section 1.61). The command REG-PROJ is an abbreviated equivalent:

```
sherpa> REGION-PROJECTION [<dataset_range> | ALLSETS] <modelname>.{<para mname> | <#>}
<modelname>.{<paramname> | <#>}
```

where <dataset_range> = #, or more generally #:#,#:#,..., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default for REGION-PROJECTION is all datasets (ALLSETS).

<**modelname**> is the name that has been given to a model component by the user. Notice that either <**paramname**> or <**#**> may be used to specify the two *thawed* parameters to be analyzed. See Chapter 1, Section 1.66 for further information regarding this command.

`REGION-UNCERTAINTY` creates a contour plot of confidence regions, computed using the `UNCERTAINTY` algorithm (See Chapter 1, Section 1.85). The command `REG-UNC` is an abbreviated equivalent:

sherpa> REGION-UNCERTAINTY [<**dataset_range**> **|** **ALLSETS**] <**modelname**>**.**{<**par amname**> **|** <**#**>} <**modelname**>**.**{<**paramname**> **|** <**#**>}

where <dataset_range> = #, or more generally #:#,#:#,. . ., such that # specifies a dataset number, and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default for `REGION-UNCERTAINTY` is all datasets (`ALLSETS`).

<**modelname**> is the name that has been given to a model component by the user. Notice that either <**paramname**> or <**#**> may be used to specify the two *thawed* parameters to be analyzed. See Chapter 1, Section 1.67 for further information regarding this command.

Examples:

1. Display 2-D data with a contour plot:

```
sherpa> READ DATA 2 example2D.dat ASCII 1 2 3
sherpa> CPLOT DATA 2
```

The last command plots dataset number 2 as a contour plot. Dataset number 2 must be a 2-D dataset.

2. Display 2-D data with a surface plot:

```
sherpa> READ DATA 2 example2D.dat ASCII 1 2 3
sherpa> SPLOT DATA 2
```

The last command plots dataset number 2 as a surface plot. Dataset number 2 must be a 2-D dataset.

3. Determine 1- and 2-sigma confidence regions for a fit:

```
sherpa> ERASE ALL
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -10
sherpa> FIT
sherpa> REGION-PROJECTION.nsigma = 2
sherpa> REGION-PROJECTION.expfac = 4
sherpa> REGION-PROJECTION my.c0 my.c2
Region-Projection: optimization reset to Levenberg-Marquardt.
                   computing grid size from covariance...done.
Region-Projection: outer grid loop 20% done...
Region-Projection: outer grid loop 40% done...
Region-Projection: outer grid loop 60% done...
```

```
Region-Projection: outer grid loop 80% done...
Minimum: 0.255412
Levels are: 2.55141 6.43641
```

4. Determine 1-, 2-, 3-, and 4-sigma confidence regions for a fit:

```
sherpa> ERASE ALL
sherpa> READ DATA example1.dat
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POLYNOM1D[my]
sherpa> THAW my.c1 my.c2
sherpa> my.c0.min = -15
sherpa> FIT
sherpa> REGION-UNCERTAINTY.nsigma = 4
sherpa> REGION-UNCERTAINTY my.c0 my.c2
Region-Uncertainty: computing grid size...done.
Region-Uncertainty: outer grid loop 20% done...
Region-Uncertainty: outer grid loop 40% done...
Region-Uncertainty: outer grid loop 60% done...
Region-Uncertainty: outer grid loop 80% done...
Minimum: 0.255412
Levels are: 2.55141 6.43641 12.0854 19.5904
```

Additional `CPLOT`, `SPLOT`, `REGION-PROJECTION`, and `REGION-UNCERTAINTY` command examples may be found in Chapter 1, Sections 1.22, 1.78, 1.66, and 1.67 respectively.

2.4.1 *ChIPS* 2-D Plot Controls

All *ChIPS* commands are available from within *Sherpa*(note, however, that axis limits on contour and surface plots may not be modified). See the *ChIPS* Reference Manual for complete *ChIPS* command definitions and further examples.

Note that *ChIPS* plot control commands require that a `REDRAW` command be issued, in order for the plot display to be updated.

Examples:

1. Plot data as a contour plot and change contour levels:

```
sherpa> DATA data/example2D.dat ASCII 1 2 3
sherpa> CPLOT DATA
sherpa> LEVELS 3 5 20 100
sherpa> REDRAW
```

   In this example, a 2-D dataset is input to *Sherpa* and then displayed as a contour plot. The *ChIPS* `LEVELS` command changes the contour levels to 3, 5, 20, and 100 for contour lines 1 thru 4, respectively.

2. Plot data as a surface plot and label the z-axis:

```
sherpa> SPLOT DATA
sherpa> ZLABEL "This is the Z Axis"
sherpa> REDRAW
```

After a surface plot has been made, a label is added to the z-axis with the *ChIPS* command `ZLABEL`.

Chapter 3

Sherpa Filtering

3.1 Introduction

Filtering within *Sherpa* may be performed using one or more of the following means:

- The `IGNORE` and/or `NOTICE` commands. With these commands, one may specify a filter interactively from a plot. These commands also support specification of a data filter using filter expressions. See Section 3.2, below.
- Input of a filter definition from a file, using the `READ` command. See Section 3.3, below.
- Input of a filtered dataset, using the `READ` command with *CIAO* Data Manipulation filtering syntax. See Section 3.4, below.

3.2 Filtering with `IGNORE | NOTICE` Commands

The `IGNORE` and `NOTICE` commands specify data-portions to be included or excluded, for 1- and 2-D datasets. The syntax for the commands is:

```
sherpa> {IGNORE | NOTICE} [#] {<arg> | FILTER <filterExpr>}
```

where `#` specifies the number of the dataset on which the command will act (default dataset number is 1). To specify an entire dataset or image, `<arg>` is `ALL`. To specify pha channels with quality flags different than 0, `<arg>` is `BAD` (this argument is valid only with the `IGNORE` command).

See Sections 1.44 and 1.55 for examples.

3.2.1 Interactive Filtering from a 1-D Plot

To interactively specify a filter region from a plot data display, the `IGNORE` or `NOTICE` command argument <`arg`> is `PLOT`:

1. Plot the data (see the `LPLOT` command, Section 1.51).
2. Issue the command `IGNORE [#] PLOT` or `NOTICE [#] PLOT`.
3. Place the cursor on the plot, at the desired minimum x-axis value for the filter, and click once with the left mouse button.
4. Place the cursor at the desired maximum x-axis value, and click again with the left mouse button. This will set the data filter to exclude or include the marked region of the dataset. Note that the character `e` may be used as an alternative to clicking the left mouse button.

Note: If you are interactively filtering on any dataset other than dataset number one, you must specify the dataset number when issuing the command `IGNORE # PLOT` or `NOTICE # PLOT`. Otherwise, the filter you specify from the data display will be applied to dataset number 1.

Please see the `LPLOT` command (Section 1.51) and Chapter 2 for further information regarding plotting capabilities within *Sherpa*.

See Sections 1.44 and 1.55 for additional examples.

Examples:

1. Exclude a 1-D data range, specified from a plot display:

   ```
   sherpa> READ DATA example.dat
   sherpa> LP DATA
   sherpa> NOTICE ALL
   sherpa> IGNORE PLOT
   <left-mouse-click once at the desired minimum x-axis value>
   <and then left-mouse-click again at the desired maximum x-axis value>
   sherpa> LP FILTER
   sherpa> LPLOT DATA
   ```

   In this example, 1-D data is first displayed. Then the data filter is set to include the entire dataset, with the command `NOTICE ALL`. The command `IGNORE PLOT` readies the cursor for selecting the desired filter region, and the user should then left-mouse-click first at the desired minimum x-axis location and then again at the maximum x-axis location. After the second left-mouse-click, the *Sherpa* command prompt is returned. The command `LPLOT FILTER` will then plot the data region that you have marked for exclusion. Finally, the dataset is plotted again, with the command `LPLOT DATA`, showing that the selected regions have been properly excluded.

3.2.2 Interactive Filtering from a 2-D Image

To interactively specify a filter region from an image data display, the `IGNORE` or `NOTICE` command argument <`arg`> is `IMAGE`:

1. Display the image and mark the filter region(s) (see the IMAGE command, Section 1.45).
2. Issue the command IGNORE [#] IMAGE or NOTICE [#] IMAGE. This will set the data filter to exclude or include the marked region of the dataset.

Note: If you are interactively filtering on any dataset other than dataset number one, you must specify the dataset number when issuing the command IGNORE # IMAGE or NOTICE # IMAGE. Otherwise, the filter you specify from the data display will be applied to dataset number 1.

Also note: If you re-display an image, following filtering, the entire image will be displayed, even though the filter has been applied to the data.

Please see the IMAGE command (Section 1.45) and Chapter 2 for further information regarding imaging capabilities within *Sherpa* .

See Sections 1.44 and 1.55 for additional examples.

Examples:

1. Exclude a 2-D region, specified from an image display:

```
sherpa> READ DATA example_img.fits
sherpa> IMAGE DATA
sherpa> NOTICE ALL
<mark regions for exclusion>
sherpa> IGNORE IMAGE
sherpa> IMAGE FILTER
```

In this example, 2-D image data is first displayed. Then the data filter is set to include the entire image, with the command NOTICE ALL. Next, the user chooses regions, by placing markers on the displayed image. The command IGNORE IMAGE sets the data filter to exclude the marked regions. The command IMAGE FILTER will display the image data regions that have been marked for exclusion.

3.2.3 1-D Filter Expressions - Syntax Option A

To specify a data filter using a filter expression, there are two different syntax options available for the IGNORE or NOTICE command argument <filterExpr>:

For Syntax Option A: This syntax is a list of data value ranges, and/or specific data values, combined using logical operators. Data value ranges are formed using either a data column name and comparison symbols, or the colon range syntax:

```
<filterExpr> = <colname>{<symbol># | =<range>} \
[<operator> <colname>{<symbol># | =<range>} ...]
```

where <colname> is the name of an appropriate data column. Note that columns must have assigned names when the data are read into *Sherpa*, in order to use this syntax option. These column names for a dataset may be viewed using the SHOW DATA command.

The following tables show the available comparison symbols, <`symbol`>, and logical operators, <`operator`>:

<symbol>	**Description**
>	greater-than
>=	greater-than or equal-to
<	less-than
<=	less-than or equal-to
!=	not equal-to
=	equal-to

Table 3.1: 1-D Filter Expression Symbols - Syntax Option A

<operator>	**Description**
&	and
^	exclusive or
|	inclusive or

Table 3.2: 1-D Filter Expression Operators - Syntax Option A

With this syntax, # is a data value being specified. Or, a range may be specified with one of the following colon range syntax options:

<range>	**Description**
value1:value2	from numerical data value1 to value2, inclusive
#:	all data values greater-than or equal-to #
:#	all data values less-than or equal-to #
#	all data values equal-to #

Table 3.3: 1-D Filter Expression Range Syntax - Syntax Option A

Examples:

1. ```
   sherpa> READ DATA "example_bin.fits[columns x1, x2]"
   sherpa> NOTICE ALL
   sherpa> IGNORE FILTER x1=4 | x1=8: | x1=1:3
   ```
   The first command inputs a data file containing named columns, and then all data points are included with `NOTICE ALL`. The filter expression then specifies column `x1` data values equal-to 4, greater-than or equal-to 8, or 1 thru 3, inclusive. This filter expression is equivalent to Example 1 of Subsection 3.2.4, when Axis 0 is named x1. The following commands are also equivalent:
   ```
   sherpa> IGNORE FILTER x1=4 | x1>=8 | x1=1:3
   sherpa> IGNORE FILTER x1=4 | x1>=8 | (x1>=1 & x1<=3)
   ```
   Note that expressions with logical operators can be grouped using parentheses, for control over logical order of operations.

2. ```
   sherpa> IGNORE ALL
   ```

```
sherpa> NOTICE FILTER x1=:2 | x1=8
```

This filter expression specifies column `x1` data values less-than or equal-to 2, or equal-to 8. This filter expression is equivalent to Example 2 of Subsection 3.2.4, when Axis 0 is x1. The command

```
sherpa> NOTICE FILTER x1<=2 | x1=8
```

is also equivalent.

3.
```
sherpa> NOTICE ALL
sherpa> IGNORE FILTER x1=:2 | x1=7:
```

This filter expression specifies column `x1` data values less-than or equal-to 2, or greater-than or equal-to 7. This filter expression is equivalent to Example 3 of Subsection 3.2.4, when Axis 0 is x1. The command

```
sherpa> IGNORE FILTER x1<=2 | x1>=7
```

is also equivalent.

4.
```
sherpa> IGNORE ALL
sherpa> NOTICE FILTER x1>5 & x1<8
```

This filter expression specifies column `x1` data values greater-than 5 and less-than 8 (*i.e.* 6 thru 7 inclusive). This filter expression is equivalent to Example 4 of Subsection 3.2.4, when Axis 0 is x1. The following commands are also equivalent:

```
sherpa> NOTICE FILTER x1=6:7
```

3.2.4 1-D Filter Expressions - Syntax Option B

To specify a data filter using a filter expression, there are two different syntax options available for the `IGNORE` or `NOTICE` command argument <filterExpr>:

For Syntax Option B: This syntax is a comma-delimited list of data value ranges, and/or specific data values, for Axis 0. Data value ranges are formed using a colon, `:`.

<**`filterExpr`**> = <**`range`**> [`,` <**`range`**>`, ...`]

where <**`range`**> is one of the following:

<`range`>	**Description**
`value1:value2`	from numerical data `value1` to `value2`, inclusive
`#:`	all data values greater-than or equal-to `#`
`:#`	all data values less-than or equal-to `#`
`#`	all data values equal-to `#`

Table 3.4: 1-D Filter Expression Range Syntax - Syntax Option B

Examples:

1. `sherpa> READ DATA "example_bin.fits[columns x1, x2]"`
   `sherpa> NOTICE ALL`
   `sherpa> IGNORE FILTER 4, 8:, 1:3`

   The first command inputs a data file containing two columns, and then all data points are included with `NOTICE ALL`. This filter expression specifies Axis 0 data values equal-to 4, greater-than or equal-to 8, or 1 thru 3, inclusive. This filter expression is equivalent to Example 1 of Subsection 3.2.3, when Axis 0 is x1.

2. `sherpa> IGNORE ALL`
   `sherpa> NOTICE FILTER :2, 8`

   This filter expression specifies Axis 0 data values less-than or equal-to 2, or equal-to 8. This filter expression is equivalent to Example 2 of Subsection 3.2.3, when Axis 0 is x1.

3. `sherpa> NOTICE ALL`
   `sherpa> IGNORE FILTER :2, 7:`

   This filter expression specifies Axis 0 data values less-than or equal-to 2, or greater-than or equal-to 7. This filter expression is equivalent to Example 3 of Subsection 3.2.3, when Axis 0 is x1.

4. `sherpa> IGNORE ALL`
   `sherpa> NOTICE FILTER 6:7`

   This filter expression specifies Axis 0 data values greater-than 5 and less-than 8 (*i.e.* 6 thru 7 inclusive). This filter expression is equivalent to Example 4 of Subsection 3.2.3, when Axis 0 is x1.

3.2.5 2-D Filter Expressions

Filter expressions for 2-D data utilize filter descriptors that are combined using logical operators. Data value ranges are formed as a comma-delimited list, contained within parantheses as part of the filter descriptor.

```
<filterExpr> = [<colname1>,<colname2> =] <descriptor> \
[<operator> <descriptor> ...]
```

where `<colname1>` and `<colname2>` are the names of appropriate data columns. Note that specifying column names is not required; if not specified, Axis 0 (`<colname1>=x1`) and Axis 1 (`<colname2>=x2`) are the defaults. The following tables show the available filter descriptors, `<descriptor>`, and logical operators, `<operator>`:

`<descriptor>`	**Description**
`BOX(x,y,width,height)`	`x` and `y` are the center coordinates of the box, `width` is twice the distance along the first axis from the center to one edge, and `height` is twice the distance along the second axis from the center to the edge.
`BOX(x,y,width,height,angle)`	`x` and `y` are the center coordinates of the box, `width` is twice the distance along the first axis from the center to one edge, `height` is twice the distance along the second axis from the center to the edge, and `angle` is the counter-clockwise rotation angle in degrees, relative to the first axis.

<descriptor>	**Description**
`CIRCLE(x,y,radius)`	`x` and `y` are the center coordinates of the circle, and `radius` is the distance from the center to the edge of the circle.
`ELLIPSE(x,y,width,height)`	`x` and `y` are the center coordinates of the ellipse, `width` is the distance along the first axis from the center to the edge, and `height` is the distance along the second axis from the center to the edge.
`ELLIPSE(x,y,width,height,angle)`	`x` and `y` are the center coordinates of the ellipse, `width` is the distance along the first axis from the center to the edge, `height` is the distance along the second axis from the center to the edge, and `angle` is the counter-clockwise rotation angle in degrees, relative to the first axis.
`POINT(x,y)`	`x` and `y` are the coordinates of a specific point.
`POLYGON(x1,y1,x2,y2,x3,y3[,...,xn,yn])`	Each `x,y` pair specifies one vertex of a polygon, up to n=32 vertices.
`PIE(x,y,start,stop)`	`x` and `y` are the apex coordinates for a filter in the shape of a pie wedge. The wedge is swept out starting at the angle `start` and stopping at the angle `stop`. The angles are given in degrees, increasing in the counter-clockwise direction, relative to the first axis.

Table 3.5: 2-D Filter Expression Descriptors

<operator>	**Description**
&	and
*	and
−	and not
∧	exclusive or
|	inclusive or
+	inclusive or

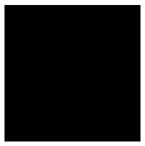
Table 3.6: 2-D Filter Expression Operators

Examples:

1. ```
   sherpa> DATA example_img.fits
   sherpa> NOTICE ALL
   sherpa> IGNORE FILTER x1,x2 = BOX(250,250,50,50)
   ```
   This filter expression specifies a box of width 50, height 50, and center at x1,x2 = 250,250. The command

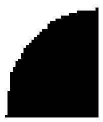
   ```
   sherpa> IGNORE FILTER BOX(250,250,50,50)
   ```

   is also equivalent.
   ```



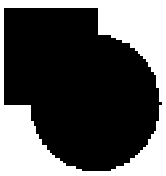
2. `sherpa> NOTICE ALL`
   `sherpa> IGNORE FILTER BOX(250,250,50,50) & CIRCLE(275,225,36)`

   This filter expression specifies an area enclosed both by the specified box and a circle of radius 36 centered at 275,225. The result is a pie wedge.



3. `sherpa> NOTICE ALL`
   `sherpa> IGNORE FILTER BOX(250,250,50,50) | CIRCLE(275,225,36)`

   This filter expression specifies an area enclosed either by the specified box or the specified circle.



4. `sherpa> NOTICE ALL`
   `sherpa> IGNORE FILTER BOX(250,250,50,50) - CIRCLE(275,225,36)`

   This filter expression specifies an area enclosed by the specified box, and not of a circle of radius 36 centered at 275,225.

3.3 Filtering with an Input Filter File

A file that contains filter assignments for each data point may be input using the `READ` command. The filter assignment value should be either 1 for inclusion, or 0 for exclusion of the data point. The syntax of the READ command for an input filter file is:

`sherpa> READ FILTER` [#] <`filespec`>

See the **`READ command`** (Section 1.64) for the <`filespec`> definition and list of accepted file formats.

Examples:

1. Input ascii data and filter files:

   ```
   sherpa> READ DATA example1.dat 1 2
   sherpa> READ FILTER example1.dat 1 3
   sherpa> SHOW FILTER
   ```

   The first command reads columns 1 and 2 of the ascii data file `example1.dat`, as dataset number 1. Column 3 of `example1.dat` contains a filter assignment for each of the data points (1 for the data point to be included; 0 for the data point to be excluded). Current filter assignments for each data point can be reported with the command `SHOW FILTER`.

3.4 Filtering Using Input Data Manipulation Syntax

The CIAO Data Manipulation (DM) software allows data to be filtered during the input/output process. As part of the CIAO software package, *Sherpa* may utilize this capability. A short description of the Data Manipulation filtering concept and syntax is given herein. For a more detailed description of the Data Manipulation software please see the *Data Manipulation Guide*.

In any CIAO software package program using the DM interface (including *Sherpa*), the user can pass a "virtual file" where an input file is expected. The virtual file is a file that is filtered (and/or binned) from an original data file, during run-time. For example, given a large 32K square HRC image, the user may wish to input only a subset of that file, with say x values within only a certain range. Using the DM filtering syntax, the user may input this filtered file, without first creating an intermediate disk file. That is, *Sherpa* will only "see" the "virtual" filtered file that has been specified. With the DM software, the virtual file is implicitly created from a true disk file by filtering or binning, but does not get created on disk or even formatted into a *UNIX* pipe. Instead, I/O calls in the program return records of the virtual file by internally accessing the actual disk file and figuring out which data passes the filter.

The full syntax for specifying a DM filter is a filename followed by a filtering and/or binning command argument, `<virtual_file_syntax>`. This command argument consists of a series of optional qualifiers within brackets, [ ]:

```
filename[block][filter][columns][binning]
```

Any qualifier my be omitted, but those which are used must appear in the order given above.

- The **block qualifier**, `[block]`: specifies which section of the file to use; in FITS, it specifies an HDU (Header Data Unit). The FTOOLS syntax of [n], where n is the HDU number, is accepted, but the name (EXTNAME value) of the HDU is equally valid and in many cases will be more intuitive. If the block qualifier is omitted, the first 'interesting' block will be used (i.e. the first FITS HDU for which NAXIS is nonzero). See Example 1, below.
- The **filter qualifier**, `[filter]`: specifies which rows of a table, or pixels of an image, to use, by filtering on the values of columns or axes. The filter on each variable is expressed as a series of acceptable ranges. Logical operators (&, <, *etc.*) may also be used. See Examples 2, 6, and 7, below.
- The **columns qualifier**, `[columns]`: specifies which columns from a table should be used; it cannot be used with images. It can also be used to rename columns, in case you have a program which expects columns with particular names. See Example 3, below.
- The **binning qualifier**, `[binning]`: makes an image from a table, by binning on any collection of columns, optionally specifying the ranges and binning factors. See Examples 4, 5, 6, and 7, below.

DM syntax may be used with the following file types:

`<filetype>`:	**Extension:**	**Description:**
FITS	fits	2-D fits image
FITSIMAGE	fits	2-D fits image
FITSBIN	fits	1-D fits binary table
IMH	imh	2-D *IRAF* image
QP	qp	2-D *PROS* qpoe

See the `READ command` (Section 1.64) for information regarding the *Sherpa* data input syntax for these file types.

Examples:

1. Input various data columns from fits binary data files:

```
sherpa> READ DATA 2 "example_bin.fits[2][cols TIME, EXPNO]" FITSBIN
```

```
sherpa> READ DATA 3 "example_bin.fits[EVENTS][cols time, expno]" FITSBIN
```

The first command reads columns `time` and `expno`, from the second extension of the fits binary table file `example_bin.fits`, as dataset number 2. The second command reads columns `time` and `expno`, from the `EVENTS` extension of the fits binary table file `example_bin.fits`, as dataset number 3. Note that columns may be specified by case insensitive name. Also, the fits extension can be specified by either the number or the name of the extension (in this example, the name of the second extension is `EVENTS`).

2. Input a portion of a 2-D fits image data file:

```
sherpa> READ DATA "example_img.fits[#1=100:200, #2=100:400]" FITS
```

This command reads the specified portion (from Axis 0 coordinates 100 to 200 and from Axis 1 coordinates 100 to 400) of the 2-D fits image `example_img.fits`. Note that, by default, the data is taken from the first fits block for which NAXIS is nonzero. The following command is equivalent:

```
sherpa> READ DATA "example_img.fits[100:200, 100:400]" FITS
```

3. Input various data columns from a fits binary data file:

```
sherpa> READ DATA 1 "example_bin.fits[2][columns #1, #2]" FITSBIN
```

This command reads the first two columns from the second extension of the fits binary table file `example_bin.fits`, as dataset number 1. *Note that column numbers or names must always be specified when reading fits binary table files.* The following commands are equivalent:

```
sherpa> READ DATA 1 "example_bin.fits[2][columns #1, #2]" FITSBIN
sherpa> READ DATA 1 "example_bin.fits[2][cols #1, #2]" FITSBIN
sherpa> DATA "example_bin.fits[2][cols #1, #2]" FITSBIN
```

4. Input a 2-D image by binning columns from a qpoe data file:

```
sherpa> DATA "example.qp[bin time=32, pha=32]" FITSIMAGE
```

This command creates and inputs into *Sherpa* an image, using the pha and time columns of of the qpoe file `example.qp`. In this example bin sizes of 32 are used, which will in this case reduce an 8192x8192 image to 256x256. Note that the `FITSIMAGE` argument is required by the *Sherpa* `READ` command syntax (since it is ultimately a fits image that is being input to *Sherpa*).

5. Input a 2-D image by binning columns from a fits binary data file:

```
sherpa> DATA "example_bin.fits[bin chipx, chipy]" FITSIMAGE
```

This command inputs into *Sherpa* the fits binary table `example_bin.fits`, but bins the table to create an image using the `chipx` and `chipy` columns. Note that the `FITSIMAGE` argument is required by the *Sherpa* `READ` command syntax.

6. Input a 2-D image by filtering and binning a column, from a fits binary data file:

```
sherpa> DATA "example_bin.fits[bin chipx=200:400:4, chipy=300:400:4]" FITSIMAGE
```

This command creates and inputs into *Sherpa*, an image using the `chipx` and `chipy` columns of the binary fits table file `example_bin.fits`. In this example, ranges for the axes, and bin sizes, are given. The resultant image size is restricted. Note that the `FITSIMAGE` argument is required by the *Sherpa* `READ` command syntax. Note however, that the following command is *not* equivalent:

```
sherpa> DATA "example_bin.fits[chipx=200:400:4, chipy=300:400:4][bin chipx, chipy]" $\backslash$ \\
FITSIMAGE
```

since this command does not restrict the overall size of the resultant image.

7. Input a 2-D image by filtering and binning columns, from a fits binary data file:

```
sherpa> READ DATA "example_bin.fits[EVENTS][time=37277062.1:37277062.2] $\backslash$ \\
[bin X=16350:16420:1,Y=16350:16420:1]" FITSIMAGE
```

This command creates and inputs into *Sherpa*, an image using the `X` and `Y` columns of the binary fits table file `example_bin.fits`. In this example, filtering is performed with two columns, for filtering both spatially and in time. The spatial subspace is the subset of the event file for X between 16350 and 16420 and Y in the same range, binned in steps of 1. The time range is from 37277062.1 to 37277062.2 seconds. Note that in this case, the order in the syntax is important. This command first goes to the events block, next filters the data in time, and then makes an image by binning over the requested X and Y region. You can't filter after binning in the same operation. In addition, once the binning is done, the data no longer contains the time information. Note that the `FITSIMAGE` argument is required by the *Sherpa* `READ` command syntax.

Chapter 4

Sherpa Optimization Methods

The primary task of *Sherpa* is to fit a model $M(\vec{p})$ to a set of observed data, where $\vec{p}$ denotes the model parameters. An *optimization method* is one that is used to determine the vector of model parameter values, $\vec{p}_o$, for which the chosen fit statistic is minimized.

Optimization is complicated because there is no guarantee that, for a given model and dataset, there will not be multiple local fit-statistic minima. Thus *Sherpa* provides a number of optimization methods, each of which has different advantages and disadvantages. The default optimization method is `POWELL`, which is a robust method for finding the nearest local fit-statistic minimum (but not necessarily the global minimum). The user can change the method using the `METHOD | SEARCHMETHOD` command (see Section 1.53). For example, to change the optimization method to `SIMPLEX`:

```
sherpa> METHOD SIMPLEX
```

Below, we provide a fuller discussion of the relative merits of different optimization methods, and then describe each of *Sherpa* optimization methods in turn. Each method section includes a description of the method, and a table listing of the method parameters. The listing of method parameters includes the *Sherpa* parameter number, and the corresponding *Sherpa* parameter name (<**paramname**>).

4.1 Advice on Minimization Methods

The contents of this section are taken from a memo written by Mark Birkinshaw (1998).

4.1.1 Introduction

The minimization of mathematical functions is a difficult operation. A general function $f(\mathbf{x})$ of vector argument $\mathbf{x}$ may have many isolated local minima, non-isolated minimum hypersurfaces, or even more complicated topologies. No finite minimization routine can guarantee to locate the unique, global, minimum of $f(\mathbf{x})$ without being fed intimate knowledge about the function by the user.

This does not mean that minimization is a hopeless task. For many problems there are techniques which will locate a local minimum which may be "close enough" to the global minimum, and there are techniques which will find the global minimum a large fraction of the time (in a probabilistic sense). However, the reader should be aware of my philosophy is that there is no "best" algorithm for finding the minimum of a general function. Instead, *Sherpa* provides tools which will allow the user to look at the overall behavior of the function and find plausible local minima ... which will often contain the physically-meaningful minimum in the types of problem with which *Sherpa* deals.

In general, the best assurance that the correct minimum has been found in a particular calculation is careful examination of the nature of the solution (e.g., by plotting a fitted function over data), and some confidence that the *full* region that the minimum may lie in has been well searched by the algorithm used. This document seeks to give the reader some information about what the different choices of algorithm will mean in terms of run-time and confidence of locating a good minimum.

Some points to take away from the discussions in the rest of this document.

1. Never accept the result of a minimization using a single optimization run; always test the minimum using a different method.
2. Check that the result of the minimization does not have parameter values at the edges of the parameter space. If this happens, then the fit must be disregarded since the minimum lies outside the space that has been searched, or the minimization missed the minimum.
3. Get a feel for the range of values of the target function (in *Sherpa* this is the fit statistic), and the stability of the solution, by starting the minimization from several different parameter values.
4. Always check that the minimum "looks right" using a plotting tool.

Sherpa contains two types of routine for minimizing a fit statistic. I will call them the "single-shot" routines, which start from a guessed set of parameters, and then try to improve the parameters in a continuous fashion, and the "scatter-shot" routines, which try to look at parameters over the entire permitted hypervolume to see if there are better minima than near the starting guessed set of parameters.

4.1.2 Single-shot techniques

As the reader might expect, the single-shot routines are relatively quick, but depend critically on the guessed initial parameter values $\mathbf{x}_0$ being near (in some sense) to the minimum $\mathbf{x}_{\rm min}$. All the single-shot routines investigate the local behaviour of the function near $\mathbf{x}_0$, and then make a guess at the best direction and distance to move to find a better minimum. After testing at the new point, they accept that point as the next guess, $\mathbf{x}_1$, if the fit statistic is smaller than at the first point, and modify the search procedure if it isn't smaller. The routines continue to run until either

1. all search directions result in an increased value of the fit statistic;
2. an excessive number of steps have been taken; or
3. something strange happens to the fit statistic (e.g., it turns out to be discontinuous in some horrible way).

This description indicates that for the single-shot routines, there is a considerable emphasis on the initial search position, $\mathbf{x}_0$, being reasonable. It may also be apparent that the values of these parameters should be moderate; neither too small (10^{-12}, say), nor too large (10^{12}, say). This is because the initial choice of step size in moving from $\mathbf{x}_0$ towards the next improved set of parameters, $\mathbf{x}_1$, is based on the change in the fit statistic, $f(\mathbf{x})$ as components of $\mathbf{x}$ are varied by amounts $\mathcal{O}(1)$. If f varies little as $\mathbf{x}$ is varied by this amount, then the calculation of the distance to move to reach the next root may be inaccurate. On the other hand, if f has a lot of structure (several maxima and minima) as $\mathbf{x}$ is varied by the initial step size, then these single-shot minimizers may mistakenly jump entirely over the "interesting" region of parameter space.

These considerations suggest that the user should arrange that the search vector is scaled so that the range of parameter space to be searched is neither too large nor too small. To take a concrete example, it would not be a good idea to have x_7 parameterize $N_{\rm H}$ in a spectral fit, with an initial guess of 10^{20}, and a search range 10^{16} to 10^{24} (units assumed to be cm^{-2}). The minimizers will look for variations in the fit statistic as $N_{\rm H}$ is varied by 1 cm^{-2}, and finding none (to the rounding accuracy likely for the code), will conclude that x_7 is close to being a null parameter and can be ignored in the fitting. It would be *much* better to have $x_7 = \log_{10}(N_{\rm H})$, with a search range of $16 - 24$. Significant variations in the fit statistic will occur as x_7 is varied by ± 1, and the code has a reasonable chance of finding a useful solution.

Bearing this in mind, the single-shot minimizers in *Sherpa* are listed below.

1. **Powell** This method is described in some detail by Press *et al.* (1986). It is basically a censored maximum-gradients technique which, starting from a first guess, moves towards a minimum by finding a good direction in which to move, and calculating a sensible distance to go. Its principal drawback is that to calculate the distance to move it has to make some assumptions about how large a step size to take, and hence there is an implicit assumption that the search space is reasonably well scaled (to ± 10 units in each of the search directions, say). It is also important that in finding these gradients, the steps do not miss a lot of important structure; there should not be too many subsidiary minima. It *is* possible for the Powell technique to become trapped in subsidiary minima, and it is possible for it to skip entirely over a minimum. However, if the initial set of parameters is not too far from the true minimum, Powell will usually do a good job of refining the parameter estimates. Thus Powell is a *good* technique for refining a set of parameters to reach a nearby local or global minimum, but a *bad* technique for finding a global minimum of a complicated search space. See also Section 4.9.

2. **Simplex** This technique creates a polyhedral search element around the initial position, $\mathbf{x}_0$, and then grows or shrinks in particular directions while crawling around parameter space, to try to place a minimum within the final search polyhedron. This technique has some hilarious ways of getting stuck in high-dimension parameter spaces (where the polyhedron can become a strange shape), but is very good at finding minima in regions where the fit statistic has a moderately well-defined topology. Since it works in a different way than Powell minimization, a good strategy is to combine Simplex and Powell minimization to test whether an apparent minimum found by one technique is stable when searched by the other. I regard Simplex searching as *good* in smooth and simple parameter spaces, particularly when looking at regions where the fit statistic depends on a parameter in a linear or parabolic fashion, and *bad* where surfaces of equal value of the fit statistic are complicated. In either case, it is essential that the initial size of the polyhedron (with sides of length 1 unit) is a smallish fraction of the search space. See also Section 4.11.

3. **Levenberg-Marquardt** This resembles an improved Powell minimization, constructing good search directions and distances to move based on the shape of the target function near the initial guessed minimum, $\mathbf{x}_0$, and then progressively moving towards the dominant local minimum. The refinement over Powell minimization is that this technique uses information about the local curvature of the fit statistic as well as its local gradients, and this tends to stabilize the result in some cases. I regard the

techniques implemented in *Sherpa* as being *good* minimum-refiners for simple local topologies, since more assumptions about topology are made than in the Simplex or Powell methods, but *bad* at finding global minima for target functions with complicated topologies. See also Section 4.5.

4.1.3 Scatter-shot techniques

Although a bit *ad hoc*, these techniques attempt to locate a decent minimum over the entire range of the search parameter space. Because they involve searching a lot of the parameter space, they involve *many* function evaluations, and are somewhere between quite slow and incredibly tediously slow.

The routines are listed below.

1. **Grid** This routine simply searches a grid in each of the search parameters. The coarseness of the grid sets how precise a root will be found. If, for example, there are 4 parameters to be searched (i.e., $\mathbf{x}$ is a vector of dimension 4), then with 131072 grid points (the default), there will be 19 sample points on each axis of the 4-D search space, and so the accuracy of estimation of each parameter will be roughly 0.06 of the range of the axis in that parameter. If the fit statistic has significant structure on a smaller scale, then the grid-searcher will miss it completely. This is a *good* technique for finding an approximation to the minimum for a slowly-varying function. It is a *bad* technique for getting accurate estimates of the location of a minimum, or for examining a fit statistic with lots of subsidiary maxima and minima within the search space. See also Section 4.3.

2. **Monte Carlo** This is even simpler than the grid-search routine. A certain number of random samples of the target function are made, with each element of the test parameter vector, $\mathbf{x}$, sampled uniformly from the specified search range. Clearly this routine is probabilistic; even for a smoothly-varying target function, there is a chance that no sample will lie particularly close to the minimum. On the other hand, for a function with unknown properties, this technique does give a chance of finding a minimum even in some part of parameter space that might not be thought of in the first place. Again, this is a *good* technique for finding rough minima for slowly-varying functions, and a *bad* technique for getting accurate minima, or minima for functions with a lot of structure in the search space. See also Section 4.6.

3. **Monte-Powell** This technique attempts to cope with complicated functions by looking for minima near a large number of randomly-selected locations within the designated search space. It turns out that this is surprisingly good at finding the global minimum of even quite complicated functions provided that enough initial start points are chosen. Difficulties may be encountered if the global minimum is in a very small part of parameter space (i.e., if the minimum is very sharp), while there are other, shallower, minima which occupy a larger fraction of the parameter space and which are likely to capture the Powell minimizations. Thus this is a *good* technique for many functions, and tends to be *bad* only for functions which are genuinely difficult anyway. It is my personal favorite for exploring a new function about which I know rather little, because it makes no special assumptions about the target function, but it is rather inefficient since many function evaluations are far from the center of parameter space. See also Section 4.8.

4. **Grid-Powell** This is similar to Monte-Powell, except that a regular grid of points is searched, rather than allowing a probabilistic sampling of the parameter space. For low-dimension spaces (*i.e.* small number of parameters) in which there is rather little structure this may be good, but for higher-dimension spaces (*i.e.* large number of parameters) the sampling of the grid becomes unacceptably coarse and finding the correct global minimum can be a considerable fluke (worse even than in Monte-Powell) because many of the function evaluations are forced to lie at the edge of the parameter space,

relatively far from the guessed minimum. Thus, for example, if an 8-D space is to be sampled, there are only 4 samples on each axis, and if the fit statistic has significant structure on the scale of a quarter of any one of the search parameters, it is likely that the global minimum will be missed. I regard Grid-Powell as a *good* method for functions where I know enough about the function that I can bracket the minimum with some certainty, but a *bad* technique for high-dimension minimizations. See also Section 4.4.

5. **Simulated Annealing** There are four such methods available in *Sherpa*. All are statistical methods for trying to find a global minimum in a complicated parameter space. The details of the various simulated annealing techniques are given in Sections 4.12 and 4.13). Basically these methods make considerable improvements on the Monte Carlo technique, by spending fewer function evaluations in unproductive bits of parameter space (where the target function is far from the minimum), and more function evaluations near the minimum. The way this is done borrows some techniques from statistical mechanics, but the essential detail is that this is a *conditioned probabilistic* approach to finding the minimum. It tends to be *good* at finding an answer which is fairly close to the global minimum, but it is *bad* in that all four the simulated annealing methods use tremendous amounts of CPU time. These techniques are a high order of overkill in relatively simple minimization problems, but for truly difficult problems where the user has little or no idea what the real answer is, they provide a brute-force way of getting an answer which is fairly reliable. The techniques that combine Powell minimization with simulated annealing tend to be better than the pure simulated annealing methods; they produce better estimates of the minima.

4.1.4 Summary and best-buy strategies

Overall, the single-shot methods are best regarded as ways of refining minima located in other ways: from good starting guesses, or from the scatter-shot techniques. Using intelligence to come up with a good first-guess solution is the best approach, when the single-shot refiners can be used to get accurate values for the parameters at the minimum. However, I would certainly recommend running at least a second single-shot minimizer after the first, to get some indication that one set of assumptions about the shape of the minimum is not compromising the solution. It is probably best if the code rescales the parameter range between minimizations, so that a completely different sampling of the function near the trial minimum is being made.

Since they tend to be slow, repeated rescaled runs are rarely possible with the scatter-shot minimizers. For most relatively well-behaved functions where quality control over the estimated parameters is being made by an intelligent user, I would recommend running Monte-Powell or Grid-Powell and then testing the minimum using the Simplex or Levenberg-Marquardt routines. If an automatic minimum-finder is being written, then I would run the scatter-shot minimizer at least twice, with the second run using a restricted range of search space around the minimum from the first, and then test the minimum using single-shot minimizers. For functions where little is known about their properties, or for problems where the user has an almost unlimited CPU time available to reach the "right" answer, I would use one of the simulated annealing plus Powell minimizers, follow up with a Grid-Powell minimizer, and then finish off with a Simplex or Levenberg-Marquardt minimizer. Even then, I wouldn't believe the solution until I'd looked at it pretty carefully: there are plenty of rogue functions out there!

Class of Technique	**Type**	**Speed**	**Commentary**
Powell	single-shot	fast	OK for refining minima
Simplex	single-shot	fast	OK for refining minima
Levenberg-Marquardt	single-shot	fast	OK for refining minima

Grid	scatter-shot	slowish	OK for smooth functions, grid-quantized minima, quad may need fine grid.
Monte Carlo	scatter-shot	slowish	OK for smooth functions, quad only gives rough minima, quad usually needs many random start points.
Grid-Powell	scatter-shot	slowish	OK for smooth functions, quad may need fine grid.
Monte-Powell	scatter-shot	slowish	OK for many functions, quad usually needs many random start points.
Simulated annealing	scatter-shot	very slow	Good in many cases, quad use if desperate measures are needed.

4.2 Summary of Available Optimization Methods

The following table lists all currently available optimization methods in *Sherpa*.

<**sherpa_methodname**>	**Description**
`GRID`	A grid search of parameter space, with no minimization.
`GRID-POWELL`	A grid search utilizing the Powell method at each grid point.
`LEVENBERG-MARQUARDT` | `LEV-MAR` | `LM`	The Levenberg-Marquardt optimization method.
`MONTECARLO`	A Monte Carlo search of parameter space.
`MONTELM`	A Monte Carlo search of parameter space, with no optimization.
`MONTE-POWELL`	A Monte Carlo search utilizing the Powell method at each selected point.
`POWELL`	The Powell optimization method.
`SIGMA-REJECTION` | `SIG-REJ`	Optimization combined with data cleansing: outliers are filtered from the data.
`SIMPLEX`	A simplex optimization method.
`SIMUL-ANN-1`	A simulated annealing search, with one parameter varied at each step.
`SIMUL-ANN-2`	A simulated annealing search, with all parameters varied at each step.
`SIMUL-POW-1`	A combination of `SIMUL-ANN-1` with `POWELL`.
`SIMUL-POW-2`	A combination of `SIMUL-ANN-2` with `POWELL`.
`USERMETHOD`	A user-defined method.

Table 4.2: Summary of Optimization Methods

Below, we describe each of these methods, in alphabetical order. Each method section includes a description of the method, and a table listing of the method parameters. The listing of method parameters includes the *Sherpa* parameter number, and the corresponding *Sherpa* parameter name (<**paramname**>).

Similar information about each method is available within *Sherpa* with the command **AHELP** <**sherpa_methodname**>, where <**sherpa_methodname**> is the name of a method.

4.3 GRID

A grid search of parameter space, with no minimization.

The `GRID` method samples the parameter space bounded by the lower and upper limits for each *thawed* parameter. At each grid point, the fit statistic is evaluated. The advantage of `GRID` is that it can provide a thorough sampling of parameter space. This is good for situations where the best-fit parameter values are not easily guessed *a priori*, and where there is a high probability that false minima would be found if one-shot techniques such as `POWELL` are used instead. Its disadvantages are that it can be slow, and that because of the discrete nature of the search, the global fit-statistic minimum can easily be missed. (The latter disadvantage may be alleviated by combining a grid search with Powell minimization; see `GRID-POWELL`.)

The user should change the parameter **`grid.totdim`** so that its value matches the number of thawed parameters in the fit. When this change is made, the total number of `GRID` parameters changes, *e.g.* if the user first changes **`grid.totdim`** to 6, the parameters **`grid.nloop05`** and **`grid.nloop06`** will appear, each with default values of 10.

If the *Sherpa* command `FIT` is given without changing **`grid.totdim`** to match the number of thawed parameters, *Sherpa* will make the change automatically. However, one cannot change the value of any newly created **`grid.nloopi`** parameters until the fit is complete.

If one is interested in running the grid only over a subset of the thawed parameter space, there are two options: first, one can freeze the unimportant parameters at specified values, or second, one can set **`grid.nloopi`** for each of the unimportant parameters to 1, while also specifying their (constant) values.

The `GRID` algorithm uses the specified minimum and maximum values for each thawed parameter to determine the grid points at which the fit statistic is to be evaluated. If **`grid.nloopi`** = 1, the grid point is assumed to be the current value of the associated parameter, as indicated above. Otherwise, the grid points for parameter x_i are given by

$$x_{i,\min}\ ,\ x_{i,\min} + \frac{(x_{i,\max} - x_{i,\min})}{(\texttt{grid.nloopi} - 1)}\ ,\ x_{i,\min} + 2\frac{(x_{i,\max} - x_{i,\min})}{(\texttt{grid.nloopi} - 1)}\ ,\ \cdots\ ,\ x_{i,\max} \tag{4.1}$$

Note that the current parameter value will not be sampled unless the minimum and maximum values, and the number of grid points, are chosen appropriately.

The maximum number of grid points that may be sampled during one fit is 1.e+7. The total number of grid points is found by taking the product of all values of **`grid.nloopi`**.

Parameters:

Number	Name	Default	Min	Max	Description
1	**`totdim`**	4	1	24	Number of free parameters.
2	**`nloopi`**	10	1	1.e+7	Number of grid points along axis i. The number of **`nloopi`** parameters which the user will actually see after issuing the command `SHOW GRID` is equal to **`totdim`**; the parameter names are **`nloop01`**, **`nloop02`**, ...

4.4 GRID-POWELL

A grid search utilizing the Powell method at each grid point.

The `GRID-POWELL` method samples the parameter space bounded by the lower and upper limits for each thawed parameter. At each grid point, the `POWELL` optimization method is used to determine the local fit-statistic minimum. The smallest of all observed minima is then adopted as the global fit-statistic minimum. The advantage of `GRID-POWELL` is that it can provide a thorough sampling of parameter space. This is good for situations where the best-fit parameter values are not easily guessed *a priori*, and where there is a high probability that false minima would be found if one-shot techniques such as `POWELL` are used instead. Its disadvantage are that it can be very slow.

Note that `GRID-POWELL` is similar in nature to `MONTE-POWELL`; in the latter method, the initial parameter value guesses in each cycle are chosen randomly, rather than being determined from a grid.

The `GRID-POWELL` method parameters are a superset of those for the `GRID` and `POWELL` methods. (See the descriptions of these methods.)

4.5 LEVENBERG-MARQUARDT — LEV-MAR

The Levenberg-Marquardt optimization method.

An abbreviated equivalent is `LEV-MAR`.

The `LEVENBERG-MARQUARDT` method is a single-shot method which attempts to find the local fit-statistic minimum nearest to the starting point. Its principal advantage is that it uses information about the first and second derivatives of the fit-statistic as a function of the thawed parameter values to guess the location of the fit-statistic minimum. Thus this method works well (and fast) if the statistic surface is well-behaved. Its principal disadvantages are that it will not work as well with pathological statistic surfaces, and there is no guarantee it will find the global fit-statistic minimum.

The code for this method is derived from the implementation in Bevington (1992).

The `eps` parameter controls when the optimization will cease; for `LEVENBERG-MARQUARDT`, this will occur when

$$|S_i - S_{i-1}| < \texttt{eps}\,, \tag{4.2}$$

where S_{i-1} and S_i are the observed statistic values for the $(i-1)^{\rm th}$ and $i^{\rm th}$ iteration, respectively.

The `smplx` parameter controls whether the `LEVENBERG-MARQUARDT` fit is refined with a `SIMPLEX` fit. `SIMPLEX` refinement can be useful for complicated fitting problems where straight `LEVENBERG-MARQUARDT` does not provide a quick solution. Switchover from `LEVENBERG-MARQUARDT` to `SIMPLEX` occurs when ΔS, the change in statistic value from one iteration to the next, is less than `LEVENBERG-MARQUARDT.smplxep`, for `LEVENBERG-MARQUARDT.smplxit` iterations in a row. For example, the default is for switchover to occur when $\Delta\chi^2 < 1$ for 3 iterations in a row.

Parameters:

Number	Name	Default	Min	Max	Description
1	`iters`	2000	1	10000	Maximum number of iterations.
2	`eps`	1.e-3	1.e-9	1	Criterion to stop fit.
3	`smplx`	1	0	1	Refine fit with simplex (0=no)
4	`smplxep`	1	0.0001	1000	Switch-to-simplex eps factor
5	`smplxit`	3	1	20	Switch-to-simplex iters factor

4.6 MONTECARLO

A Monte Carlo search of parameter space.

The `MONTECARLO` method randomly samples the parameter space bounded by the lower and upper limits of each *thawed* parameter. At each chosen point, the fit statistic is evaluated. The advantage of `MONTECARLO` is that it can provide a good sampling of parameter space. This is good for situations where the best-fit parameter values are not easily guessed *a priori*, and where there is a high probability that false minima would be found if one-shot techniques such as `POWELL` are used instead. Its disadvantages are that it can be slow (if many points are selected), and that because of the random, discrete nature of the search, the global fit-statistic minimum can easily be missed. (The latter disadvantage may be alleviated by combining a Monte Carlo search with Powell minimization; see `MONTE-POWELL`.)

If the number of thawed parameters is larger than 3, one should increase the value of `nloop` from its default value. Otherwise the sampling may be too sparse to estimate the global fit-statistic minimum well.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	500	1	1.6777e+7	Number of parameter space samples.
2	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.

4.7 MONTE-LM

A Monte Carlo search utilizing the Powell method at each selected point.

The `MONTE-LM` method randomly samples the parameter space bounded by the lower and upper limits for each *thawed* parameter. At each grid point, the `LEVENBERG-MARQUARDT` optimization method is used to determine the local fit-statistic minimum. The smallest of all observed minima is then adopted as the global fit-statistic minimum. The advantage of `MONTE-LM` is that it can provide a good sampling of parameter space. This is good for situations where the best-fit parameter values are not easily guessed *a priori*, and where there is a high probability that false minima would be found if one-shot techniques such as `LEVENBERG-MARQUARDT` are used instead. Its disadvantage is that it can be slow.

The `MONTE-LM` method parameters are a superset of those listed for the `LEVENBERG-MARQUARDT` method and the ones listed below.

If the number of thawed parameters is larger than 2, one should increase the value of `nloop` from its default value. Otherwise the sampling may be too sparse to estimate the global fit-statistic minimum well.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	128	1	16384	Number of parameter space samples.
2	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.

4.8 MONTE-POWELL

A Monte Carlo search utilizing the Powell method at each selected point.

The `MONTE-POWELL` method randomly samples the parameter space bounded by the lower and upper limits for each *thawed* parameter. At each grid point, the `POWELL` optimization method is used to determine the local fit-statistic minimum. The smallest of all observed minima is then adopted as the global fit-statistic minimum. The advantage of `MONTE-POWELL` is that it can provide a good sampling of parameter space. This is good for situations where the best-fit parameter values are not easily guessed *a priori*, and where there is a high probability that false minima would be found if one-shot techniques such as `POWELL` are used instead. Its disadvantage is that it can be very slow.

Note that `MONTE-POWELL` is similar in nature to `GRID-POWELL`; in the latter method, the initial parameter values in each cycle are determined from a grid, rather than being chosen randomly.

The `MONTE-POWELL` method parameters are a superset of those listed for the `POWELL` method and the ones listed below.

If the number of thawed parameters is larger than 2, one should increase the value of `nloop` from its default value. Otherwise the sampling may be too sparse to estimate the global fit-statistic minimum well.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	128	1	16384	Number of parameter space samples.
2	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.

4.9 POWELL

The Powell optimization method.

The `POWELL` method is a single-shot method which attempts to find the local fit-statistic minimum nearest to the starting point. Its principal advantage is that it is a robust *direction-set* method. A set of directions (*e.g.*, unit vectors) are defined; the method moves along one direction until a minimum is reached, then from there moves along the next direction until a minimum is reached, and so on, cycling through the whole set of directions until the fit statistic is minimized for a particular iteration. The set of directions is then updated and the algorithm proceeds. Its principal disadvantages are that it will not find the local minimum as quickly as `LEVENBERG-MARQUARDT` if the statistic surface is well-behaved, and there is no guarantee it will find the global fit-statistic minimum.

The **eps** parameter controls when the optimization will cease; for `POWELL`, this will occur when

$$|S_i - S_{i-1}| < \frac{1}{2} * \texttt{eps} * (|S_i| + |S_{i-1}|) \,, \tag{4.3}$$

where S_{i-1} and S_i are the observed statistic values for the $(i-1)^{\rm th}$ and $i^{\rm th}$ iteration, respectively.

Parameters:

Number	Name	Default	Min	Max	Description
1	`iters`	2000	1	10000	Maximum number of iterations.
2	`eps`	1.e-6	1.e-9	0.001	Criterion to stop fit.
3	`tol`	1.e-6	1.e-8	0.1	Tolerance in lnmnop
4	`huge`	1.e+10	1000	1.e+12	Vestigial.

4.10 SIGMA-REJECTION — SIG-REJ

The SIGMA-REJECTION optimization method for fits to 1-D data.

Abbreviated equivalents are SIG-REJ and SR.

The `SIGMA-REJECTION` optimization method is based upon the *IRAF* method `SFIT`. It iteratively fits data: a best-fit to the data is determined using a 'regular' optimization method (*e.g.* `LEVENBERG-MARQUARDT`), then outliers data points are filtered out of the dataset, and the data refit, *etc.* Iterations cease when there is no change in the filter from one iteration to the next, or when the fit has iterated a user-specified maximum number of times.

Parameters:

Number	Name	Default	Min	Max	Description
1	`niter`	5	0	128	The maximum number of iterations in the fit; if 0, the fit will run to convergence (*i.e.*, until there is no change in the filter).
2	`lrej`	3	0	10	Data point rejection criterion in units of *sigma*, for data points *below* the model (*e.g.*, absorption line data); using `lrej` is equivalent to doing LPLOT DELCHI and filtering out all data points lying below $y = -$lrej on that plot.
3	`hrej`	3	0	10	Data point rejection criterion in units of *sigma*, for data points *above* the model (*e.g.*, absorption line data); using `hrej` is equivalent to doing LPLOT DELCHI and filtering out all data points lying above $y =$ hrej on that plot.

4	`grow`	0	0	1024	When a given data point is to be filtered out, this parameter sets the number of pixels adjacent to that pixel which are also to be filtered out, *i.e.*, if 0, only the data point itself is filtered out; if 1, the data point and its two immediate neighbors are filtered out, *etc.*
5	`omethod`	lm			The optimization method to use to perform fits at each iteration. Note that quotes are required around the name!

4.11 SIMPLEX

A simplex optimization method.

The `SIMPLEX` method is a single-shot method which attempts to find the local fit-statistic minimum nearest to the starting point. Its principal advantage is that it *can* work well with complicated statistic surfaces (more so than `LEVENBERG-MARQUARDT`), while also working quickly (more so than `POWELL`). Its principal disadvantages are that it has a tendency to "get stuck" in regions with complicated topology before reaching the local fit-statistic minimum, and that there is no guarantee it will find the global fit-statistic minimum. Its tendency to stick means that the user may be best-served by repeating fits until the best-fit point does not change.

A simplex is geometrical form in N-dimensional in parameter space which has $N + 1$ vertices (*e.g.*, in 3-D it is a tetrahedron). The fit statistic is evaluated for each vertex, and one or more points of the simplex are moved, so that the simplex moves towards the nearest local fit-statistic minimum. When a minimum is reached, the simplex may also contract itself, as an amoeba might; hence, the routine is also sometimes called "amoeba." Convergence is reached when the simplex settles into a minimum and all the vertices are within some value `eps` of each other.

The `eps` parameter controls when the optimization will cease; for `SIMPLEX`, this will occur when

$$|S_i - S_{i-1}| < \texttt{eps}\,, \tag{4.4}$$

where S_{i-1} and S_i are the observed statistic values for the $(i-1)^{\text{th}}$ and i^{th} iteration, respectively.

Parameters:

Number	**Name**	**Default**	**Min**	**Max**	**Description**
1	`iters`	2000	1	10000	Maximum number of iterations.
2	`eps`	1.e-6	1.e-9	0.001	Criterion to stop fit.
3	`alpha`	1	0.1	2	Algorithm convergence factor.
4	`beta`	0.5	0.05	1	Algorithm convergence factor.
5	`gamma`	2	1.1	20	Algorithm convergence factor.

4.12 SIMUL-ANN-1

A simulated annealing search, with one parameter varied at each step.

The `SIMUL-ANN-1` method is a simulated annealing method. Such a method is said to be useful when a desired global minimum is among many other local minima. It has been derived via an analogy to how metals cool and anneal; as the liquid metal is cooled slowly, the atoms are able to settle into their minimum energy state, *i.e.* form a crystal. In simulated annealing, the function to be optimized is held to be the analog of energy, and the system slowly "cooled." At each temperature tried, the parameters are randomly varied; when there is no further improvement, the temperature is lowered, and again the parameters are varied some number of times. With sufficiently slow cooling, and sufficient random tries, the algorithm "freezes" near the global minimum. (At each randomization, only one parameter is varied.) This is quite different from the other techniques implemented here, which can be said to head downhill, fast, for a nearby minimum. The great advantage of simulated annealing is that it sometimes goes uphill, thus escaping undesirable local minima the other techniques tend to get stuck in. The great disadvantage, of course, is that it is a slow technique.

A simple view of simulated annealing is that a new guess at the location of the minimum of the objective function is acceptable if it causes the value of the function to decrease, and also with probability

$$P = \exp\left(-\frac{\Delta f}{T}\right) \tag{4.5}$$

where Δf is the *increase* in the value of the objective function, and T is a "temperature." After a large number of random points in parameter space have been sampled, and no improvement in the minimum value of the objective function has occurred, the temperature is decreased (increasing the probability penalty for positive Δf), and the process is repeated. The expectation is that with a sufficient number of sampled points at each temperature, and a sufficiently slow "cooling" of the system, the result will "freeze out" at the true minimum. The analogy is supposed to be with a cooling gas, where the Boltzmann statistics of molecular motions causes the molecules of the gas to explore both the minimum energy state and adjacent energy states at any one temperature, and then freeze in a crystalline state of minimum energy as the temperature approaches zero.

`SIMUL-ANN-1` starts with a temperature which is about twice the initial value of the objective function, and reduces it slowly towards zero, changing a single parameter value at each randomization.

Parameter `tchn` is a multiplying factor for the temperature variable between cooling steps. Its default value of 0.98 means that a large number of cooling steps is needed to reduce the temperature sufficiently that the function "freezes out," but fast cooling is not recommended (as it inhibits the ergodic occupation of the search space). Parameter `nloop` specifies the maximum number of temperature values to try–but may be overridden if the code finds that the solution has "frozen out" early. Finally, parameter `nsamp` specifies the number of movements at each temperature–the number of random changes to the parameter values that will be tried.

`SIMUL-ANN-1` tends to be slow, but is a possible way of making significant progress in traveling-salesman type problems. One of its strengths is that even if it doesn't find the absolutely best solution, it often converges to a solution that is close (in the Δf sense) to the true minimum solution. Somewhat better results are often obtained by `SIMUL-POW-1`, Section 4.14 and `SIMUL-POW-2`, Section 4.15, where the simulated annealing method is combined with the Powell method.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	512	32	8192	Maximum number of temperatures.
2	`tchn`	0.98	0.1	0.9999	Factor for temperature reduction.
3	`nsamp`	256	16	4096	Number of movements at each temperature.
4	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.
5	`tiny`	1.e-12	1.e-20	1.e-6	Smallest temperature allowed.

4.13 SIMUL-ANN-2

A simulated annealing search, with all parameters varied at each step.

The `SIMUL-ANN-2` method is a simulated annealing method, analogous to `SIMUL-ANN-1` in every way except that at each randomization, all the parameters are varied, not just one.

Note that the default temperature cycle change factor, and the number of temperature cycles, correspond to more and faster changes of temperature than `SIMUL-ANN-1`. This is because of the increased mobility within parameter space, since all values are changed with each randomization.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	1024	64	16384	Maximum number of temperatures.
2	`tchn`	0.95	0.1	0.9999	Factor for temperature reduction.
3	`nsamp`	512	32	8192	Number of movements at each temperature.
4	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.
5	`tiny`	1.e-12	1.e-20	1.e-6	Smallest temperature allowed.

4.14 SIMUL-POW-1

A combination of `SIMUL-ANN-1` with `POWELL`.

This method packages together `SIMUL-ANN-1` and the `POWELL` routine; at the end of each of the cooling sequences, or annealing cycles, the `POWELL` method is invoked. Probably one of the best choices where one 'best' answer is to be found, but at the expense of a lot of computer time.

Note that the parameters of `SIMUL-POW-1` are those of `SIMUL-POW-1` itself (which have the same meaning as in routine `SIMUL-ANN-1`), *plus* those of `POWELL`.

All of the `SIMUL-POW-1` parameters, with the exception of `nanne`, are explained under `SIMUL-ANN-1`, Section 4.12. Parameter `nanne` specifies the number of annealing cycles to be used; successive anneal cycles start from cooler temperatures and have slower cooling. The pattern of anneals that has been chosen (the "annealing history") is not magic in any way–for best results a different annealing history may be better for any particular objective function–but the pattern chosen seems to serve well.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	256	16	4096	Maximum number of temperatures.
2	`tchn`	0.95	0.1	0.9999	Factor for temperature reduction.
3	`nanne`	16	1	256	Number of anneals.
4	`nsamp`	128	16	1024	Number of movements at each temperature.
5	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.
6	`tiny`	1.e-12	1.e-20	1.e-6	Smallest temperature allowed.

4.15 SIMUL-POW-2

A combination of SIMUL-ANN-2 with POWELL.

This method packages together `SIMUL-ANN-2` and the `POWELL` routine; at the end of each of the cooling sequences, or annealing cycles, the `POWELL` method is invoked. The rate of cooling in each anneal loop may be much faster than in simulated annealing alone. Probably one of the best choices where one 'best' answer is to be found, but at the expense of a lot of computer time.

Note that the parameters of `SIMUL-POW-2` are those of `SIMUL-POW-2` itself (which have the same meaning as in routine `SIMUL-ANN-2`), *plus* those of `POWELL`.

All of the `SIMUL-POW-2` parameters, with the exception of `nanne`, are explained under `SIMUL-ANN-2`, Section 4.13. Parameter `nanne` specifies the number of annealing cycles to be used; successive anneal cycles start from cooler temperatures and have slower cooling. The pattern of anneals that has been chosen (the "annealing history") is not magic in any way–for best results a different annealing history may be better for any particular objective function–but the pattern chosen seems to serve well.

Parameters:

Number	Name	Default	Min	Max	Description
1	`nloop`	256	16	4096	Maximum number of temperatures.
2	`tchn`	0.95	0.1	0.9999	Factor for temperature reduction.
3	`nanne`	16	1	256	Number of anneals.
4	`nsamp`	128	16	1024	Number of movements at each temperature.
5	`iseed`	14391	-1.e+20	1.e+20	Seed for random number generator.
6	`tiny`	1.e-12	1.e-20	1.e-6	Smallest temperature allowed.

4.16 USERMETHOD

A user-defined method.

It is possible for the user to create and implement his or her own model, own optimization method, and own statistic function within *Sherpa*. The *User Models, Statistics, and Methods Within Sherpa* chapter (Chapter 7) of the *Sherpa Reference Manual* has more information on this topic.

The tar file **sherpa_user.tar.gz** contains the files needed to define the **usermethod**, e.g Makefiles and Implementation files, plus example files, and it is available from the Sherpa threads page: Data for Sherpa Threads

Chapter 5

Sherpa Models

5.1 Introduction

The primary task of *Sherpa* is to fit a *model* $M(\vec{x}, \vec{p}, O)$ to a set of observed data. $\vec{x}$ denotes the set of model *arguments*, $\vec{p}$ denotes the model *parameters*, and O denotes model *options*.

1. Arguments are generic to the model, and describe the axis along which the model fit occurs. For example, when fitting energy spectra, $\vec{x}$ could be a set of photon-energy bin boundaries defined in units of keV.

2. Parameters control the model's shape as a function of the model arguments. For example, a and b are parameters that control the amplitude and slope of the power-law model ax^{-b}. Parameters can be varied during the fitting process until their best-fit values are found, or held fixed to specified values.

3. Options place special constraints on the model. For example, in an absorption model the opacity can be input from different tables, so while the mathematical form of the model function remains the same, the computed values of M can vary. Options remain constant during the fitting process.

The modeling aspect of *Sherpa* is designed with complete generality in mind. *Sherpa* provides the user with a set of commonly used 1- and 2-D models, some of which are unique to *Sherpa*, and some of which come directly from Version 11 of *XSPEC*(Arnaud 1996). The user is free to combine these models to build arbitrarily complex composite models.[1] The user can also rename models, and apply them along any data axis. Thus the user can provide three names for the 1-D power-law model `POWLAW1D` and build a 3-D composite power-law model with different photon indices along each axis. (The photon index values could also be linked between different axes.) Finally, *Sherpa* also provides the user with the capability of inputting his or her own model (see `USERMODEL` below).

[1]The majority of the models in *XSPEC*are 1-D, created to fit energy spectra (in units of keV). Therefore the user must exercise caution when combining models from the *Sherpa* and *XSPEC*libraries—the models must be a function of the same arguments. For instance, data may be input to *Sherpa* with any units, so if the user wishes to combine models, the data should be defined in keV, or the user should supply an instrument model that provides a mapping from the observed units to keV. Instrument models are discussed later in this chapter.

There are three model categories in *Sherpa*: source, background, and instrument. Each is discussed in turn below. Models from different categories can be combined during the fitting process. The user can combine background and source models when fitting background-subtracted data, or convolve source and instrument models when fitting PSF-broadened spatial data. *Sherpa* does not require the user to specify models in all three categories; only the source model is required.

5.2 Source Models

Sherpa uses the concept of '*source models*' in order to highlight their specific physical meaning. They describe parent distributions from which the observed source data may have been sampled. While many of the *Sherpa* source models have physical interpretation, many are phenomenological, serving only as a mathematical parameterization of the observed data.

Source models may be compared to the observed data in two different ways:

- The user can elect to directly compare source models with the observed data, if for instance they are dominated by the source signal, and the effect of instrument characteristics is negligible. For example, the user may elect to model optical emission lines directly with simple 1-D Gaussian functions.
- The user can also elect to convolve source models with an instrumental response function, then compare the result with the observed data, if for instance they are dominated by instrumental features. For example, the standard analysis of X-ray energy spectra requires source models to be convolved with the response matrix of the instrument. The model predicted counts, $M(i_{\rm PHA})$, are then given by

$$M(i_{\rm PHA}) = \int_{E_{\rm lo}}^{E_{\rm hi}} R(i_{\rm PHA}, E) f(E) dE \tag{5.1}$$

where $R(i_{\rm PHA}, E)$ is the instrument response,[2] which describes the probability that a photon with energy $E_{\rm lo} \leq E \leq E_{\rm hi}$ will be detected as a count in the PHA channel $i_{\rm PHA}$, and where $f(E)$ is the specified source model function. Another example would be the convolution of a 2-D spatial source model with an instrumental Point-Spread Function (PSF).

5.2.1 1-D Source Models

Sherpa provides a number of 1-D source models that the user may apply to any kind of 1-D data, or may combine together to create arbitrarily complex source models. For instance, one could use `POWLAW1D` to model energy spectra, `SIN` to model a binned time series (e.g. a light curve), and `GAUSS` to arbitrary non-astrophysical data provided in an ASCII file.

5.2.2 2-D Source Models

Sherpa also provides a number of 2-D source models that the user may apply to any kind of 2-D data, or may combine together to create arbitrarily complex source models. For instance, one could use `BETA2D` or

[2] R may include contributions that are defined in separate Response Matrix Format [RMF] and Ancillary Response Format [ARF] files. [NEED REFERENCE HERE.]

`GAUSS2D` to fit image data. These 2-D models are sometimes referred to as spatial models in this manual, since it is envisioned that their use will be primarily to fit 2-D image data.

5.2.3 Multi-dimensional Source Models

Chandra is capable of making simultaneous high resolution measurements of a photon's spatial location, energy, and time-of-arrival. Thus the user may wish to combine 1-D and/or 2-D models to create arbitrarily complex 2-D, 3-D, or 4-D models to fit these data. Examples of analyses which require such multi-dimensional models include:

- Supernova Remnants: the user may wish to fit a spectrum in each bin on a 10 arcsec grid, make arrays of Fe-K and Si-K equivalent widths, plot the ratio on a map, and plot ratio versus temperature.
- Cooling flow: the user may wish to create a 3-D flow model and project it to 2-D, and fit to surface brightness, while also modeling the energy spectrum.
- Energy spectrum versus time: the user may wish to make a 2-D image of the energy spectrum versus time, describing temporal variations in the spectrum.
- Aspect quality: the user may wish to divide the data into images binned by, e.g. guide star separation, angle to Earth's B-field, optical bench temperature, *etc.*, and measure image FWHM in each bin and plot FWHM versus B-field.

As stated above, *Sherpa* allows the user to combine 1-D models so as to model a 2-D dataset (with, say, energy and time axes). Note however that in the current version of *Sherpa* it is not yet possible to combine 1-D and/or 2-D instrument models, i.e. the user cannot yet convolve an arbitrary multi-dimensional model with a multi-dimensional instrument model. Such functionality will be provided in a future version of the software.

5.2.4 Source Model Integration

In the analysis of binned data, source model functions can be split into two classes. The first class contains the "additive" models, which are integrated over bins in energy-space (before being folded through an instrument), or over bins in counts-space (if no instrument model is specified). The second class contains the "multiplicative" models, which are not integrated when evaluated. Additive models can describe photon generation (e.g. `BBODY`), while multiplicative models can describe photon attenuation (e.g. `ATTEN`) or filtering (e.g. `BOX1D`). Note that in *Sherpa*, multiplicative model amplitudes for binned data are currently evaluated at the lower boundary of each bin.

In *Sherpa*, some models are integrated by default, including all *XSPEC* additive models (see Table 5.9 and the *XSPEC* Version 11 User's Guide). *Sherpa* lets the user control the integration of non-*XSPEC* source models (i.e. those models unique to *Sherpa*) when fitting binned data, by using the `INTEGRATE` <`ON|OFF`> command (Section 5.4.13). An example of a situation in which the user may wish to change the integration status of a source model is when it is used in nested model expressions.

5.3 Background Models

Background models are in every way like source models, except that they are used to describe parent distributions from which observed background data may have been sampled. These data are input using the `BACK` command (Section 1.7). Depending on the type of statistical analysis, these data can be subtracted from the observed raw counts data or can be simultaneously modeled with the source data. Background models are specified using the `BACKGROUND | BG` command (Section 1.9). note that any model component in the *Sherpa* library may be used to compose a background model.

5.4 Instrument Models

Instrument models describe instrument characteristics, such as an effective area, a detector's energy response, or a mirror's point-spread function. They are convolved with the source, background, or combined source and background model to compute the number of predicted counts in each detector bin.

Instrument models may also be defined with analytic functions. The current version of *Sherpa* allows the user to define instrument models using functions defined in the *TCD* library (the *T*ransformation, *C*onvolution and *D*econvolution tools). These include 1-D and 2-D Gaussian, box, and top-hat functions. The relevant command is `TPSF`.

The `INSTRUMENT` command (Section 1.46) has more details on modeling the instrument in source or background data analysis.

The following instrument models are available in *Sherpa* 3.2:

Model	**Description**
`FARF1D | FARF`	A 1-D file-based ancillary response model
`FARF2D | FEXPMAP | FEXPMAP2D`	A 2-D file-based ancillary response model
`FPSF1D`	A 1-D file-based PSF instrument model
`FPSF2D | FPSF | PSFFROMFILE`	A 2-D file-based PSF instrument model
`FRMF1D | FRMF`	A 1-D file-based response matrix model
`KERNEL`	A 1-D convolution model.
`PTSRC1D`	A 1-D file-based point-source fitting model
`PTSRC2D | PTSRC`	A 2-D file-based point-source fitting model
`RSP1D | RSP`	A 1-D instrument model.
`RSP2D`	A 2-D instrument model.
`TPSF1D`	A 1-D TCD-model-based PSF instrument model
`TPSF2D | TPSF |PSFFROMTCD | PSF`	A 2-D TCD-model-based PSF instrument model

Table 5.1: Summary of Instrument Models

More information on each model is available at the end of this chapter.

5.4.1 Establishing Model Components

The following syntax establishes a model component for usage in the current *Sherpa* session:

`sherpa> [CREATE] <sherpa_modelname>`

(The brackets around `CREATE` indicate that its use is optional; see examples below.) In addition, *Sherpa* provides the user with the ability to both establish a model component, and to assign to it an arbitrary name:

`sherpa> [CREATE] <sherpa_modelname>[<modelname>]`

where <**sherpa_modelname**> is the *Sherpa* default model name (or an *XSPEC* model name, with the prefix 'xs' attached), and <**modelname**> is the name being given to the model component by the user. Note that <**modelname**> *must be enclosed in brackets*, [ ].

Sherpa's ability to assign arbitrary names to model components allows the user to establish multiple independent models of the same type during a single session, and is a valuable feature of the software. (See Example 7.) Note that an assigned model name can be any arbitrary string, except a string that is already a *Sherpa* command.

By default, *Sherpa* will prompt the user for the initial model component parameter values.[3] At the model parameter prompt, the user may either:

- Enter the <RETURN> `key` to accept the given initial parameter value. Note that if a dataset has not been input, then the initial parameter values are arbitrary default values. However, if a dataset has been input, then the initial parameter values are estimates based on the filtered data and the defined instrument model (see Section 5.4.2 for more information).
- Enter the <RETURN> `key` to accept the given initial parameter value, and subsequently set the parameter value and range using one of the syntax options described in Sections 5.4.3, 5.4.4, and 5.4.5.
- Enter the <RETURN> `key` to accept the given initial parameter value, and subsequently set the parameter value using the `GETX` or `GETY` command (see Sections 1.39 and 1.40), which allows the user to assign the value interactively from a plot.
- Enter `,-1` to accept the given initial parameter value, and to set the parameter to be frozen. Note that there *cannot* be a space between the `,` and the `-1`.
- Enter <`value`>`:`<`min`>`:`<`max`>[`:`<`delta`>][`,-1`], where:

Argument:	**Description:**
<`value`>	Model parameter initial value.
<`min`>	Minimum value for that parameter range.
<`max`>	Maximum value for that parameter range.
<`delta`>	Specifies initial parameter step size.
`,-1`	Sets the parameter to be frozen.

Note that the default setting for `delta` is 1% of the parameter value. Otherwise, specifying `delta` will set the initial parameter step size to an absolute value. The parameter step size is used by

[3] Parameter prompting can be turned off using the command `PARAMPROMPT OFF` (see Section 1.58).

the optimization method to determine where to sample parameter space. As such, if the value of a parameter is known well, then specifying a small `delta` may lead to a substantially faster and better fit.

Note that this colon separated list need not include all of the elements, but it does need to maintain the proper sequential order, and also include the colons around skipped elements (up to three elements only). Also, there cannot be a space between the `,` and the `-1`.

The model components that have been established in the current *Sherpa* session, and their parameter information, may be listed with the command `SHOW MODELS`. Note however that information about model parameter `delta` settings is currently not returned by `SHOW`. Information about model parameter `delta` settings is included when using `SAVE`.

Examples:

1. Establish a model component:

```
sherpa> CREATE POLY
POLY.c0 parameter value [1]
POLY.c1 parameter value [0]
POLY.c2 parameter value [0]
POLY.c3 parameter value [0]
POLY.c4 parameter value [0]
POLY.c5 parameter value [0]
POLY.c6 parameter value [0]
POLY.c7 parameter value [0]
POLY.c8 parameter value [0]
POLY.offset parameter value [0]
```

The command `CREATE POLY` establishes the *Sherpa* model `POLYNOM1D` as a model component available for usage in the current *Sherpa* session. Note that after issuing this command, the user is prompted for the initial model parameter values. In this example, the user accepted the given initial values for all of the parameters, using the `<RETURN> key`. The following command is equivalent:

```
sherpa> POLY
```

2. Establish a model component, and assign it a name:

```
sherpa> CREATE POLY[modela]
modela.c0 parameter value [1]
modela.c1 parameter value [0]
modela.c2 parameter value [0]
modela.c3 parameter value [0]
modela.c4 parameter value [0]
modela.c5 parameter value [0]
modela.c6 parameter value [0]
modela.c7 parameter value [0]
modela.c8 parameter value [0]
modela.offset parameter value [0]
```

The command `CREATE POLY[modela]` establishes the *Sherpa* model component `POLYNOM1D`, and assigns to it the name `modela`. Note that after issuing this command, the user is prompted for the initial model parameter values. In this example, the user accepted the given initial values for all of the parameters, using the <RETURN> `key`. The following command is equivalent:

```
sherpa> POLY[modela]
```

3. Establish a model component, assign it a name, set parameter values and ranges including initial parameter step size, and freeze a parameter:

```
sherpa> GAUSS[modelb]
modelb.fwhm parameter value [10] 2.0:1:10:0.5
modelb.pos parameter value [0] 1:0.1:
modelb.ampl parameter value [1] 3::100:2,-1
```

This example establishes and assigns the name `modelb` to the *Sherpa* model component `GAUSS1D`. Note that in this example the user entered an initial value of 2.0, a minimum range value of 1, a maximum range value of 10, and an initial parameter step size `delta` of 0.5 for parameter `fwhm`; entered a value of 1, and a minimum range value of 0.1 for parameter `pos`; and entered a value of 3, a maximum range value of 100, and an initial parameter step size of 2 for parameter `ampl`, and froze this parameter.

4. Establish a model component, assign it a name, set parameter values and ranges, and freeze a parameter:

```
sherpa> GAUSS[modelbb]
modelbb.fwhm parameter value [10] 2.0:1:10
modelbb.pos parameter value [0] 1:0.1:
modelbb.ampl parameter value [1] 3::100,-1
```

This example establishes and assigns the name `modelbb` to the *Sherpa* model component `GAUSS1D`. Note that in this example the user entered an initial value of 2.0, a minimum range value of 1, and a maximum range value of 10 for parameter `fwhm`; entered a value of 1, and a minimum range value of 0.1 for parameter `pos`; and entered a value of 3, and a maximum range value of 100 for parameter `ampl`, and froze this parameter (note that after the maximum range value, specification of initial parameter step size `delta` is optional and may be omitted).

5. Establish an *XSPEC*model component, assign it a name, and freeze a parameter:

```
sherpa> XSBBODY[modeld]
modeld.kT parameter value [3]
modeld.norm parameter value [1] ,-1
```

This command establishes and assigns the name `modeld` to the *XSPEC*model component `bbody`. In this example, the user accepted the given initial values for all of the parameters, using the <RETURN> `key`, and set parameter `norm` to be frozen (see Section 5.4.11 for further information about freezing and thawing parameters).

6. Turn off model parameter prompting; establish a model component and assign it a name:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
```

The first command turns off prompting for model parameter values. The second command establishes the *Sherpa* model component `POWLAW1D`, and assigns to it the name `modelc`. Since parameter prompting was turned off, the model is automatically established using the given initial values for all of the parameters.

7. Establish two independent model components of the same type:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> POW[modelc]
modelc.gamma parameter value [0] 1.0
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
sherpa> POW[modelc2]
modelc2.gamma parameter value [0] 2.0
modelc2.ref parameter value [1]
modelc2.ampl parameter value [1]
```

The third command establishes the *Sherpa* model component `POWLAW1D`, and assigns to it the name `modelc`. In this example, the user set parameter `gamma` of model component `modelc` to the value of 1.0. The last command establishes another *Sherpa* model component `POWLAW1D`, and assigns to it the name `modelc2`. In this example, the user set parameter `gamma` of model component `modelc2` to the value of 2.0. Note that model components `modelc` and `modelc2` are independent of one another:

```
sherpa> SHOW MODELS
Current Composite Models are:

Current Model Components are:
powlaw1d[modelc]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          1        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1     0.0000 3.4028e+38

powlaw1d[modelc2]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          2        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1     0.0000 3.4028e+38
```

5.4.2 Automatic Estimates of Initial Parameter Values

It is important to note that when a model component is established, if a dataset has been input, then estimates based on the (filtered) data are automatically made for the initial parameter values and their minimum and maximum ranges. However, if no dataset has been input, then the initial parameter values and ranges are assigned arbitrary default values.

If multiple datasets have been input, and if no dataset is indicated in the model expression, then automatic parameter estimates are based on the lowest numbered dataset; for instance, if the user inputs

```
sherpa> DATA 1 example1.dat
sherpa> DATA 2 example2.dat
sherpa> POLY[modela]
```

then the estimates of `modela`'s parameter values are based on the first dataset. Otherwise, if multiple datasets have been input, and a dataset is indicated in the model expression, then the automatic parameter estimates are made based on the indicated dataset; for instance, if the user inputs

```
sherpa> DATA 1 example1.dat
sherpa> DATA 2 example2.dat
sherpa> SOURCE 2 = POLY[modela]
```

then the estimates of `modela`'s parameter values are based on the second dataset.

The user may always obtain updated estimates using the command `GUESS` (see Section 1.43) (see Example 3 of Section 1.43). Updated estimates may be needed if:

- a dataset is input,
- or an instrument model is specified,
- or a dataset filter is changed,
- or a dataset WCS analysis setting is changed,

after a model expression is input.

The user may of course wish to directly set initial parameter values and ranges; see Sections 5.4.1, 5.4.3, 5.4.4, and 5.4.5.

Examples:

1. Establish a model component prior to data input:

```
sherpa> ERASE ALL
sherpa> POW[modelc]
modelc.gamma parameter value [0]
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
sherpa> SHOW modelc
powlaw1d[modelc]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1     0.0000 3.4028e+38
```

Here, a model component is established prior to input of data, so the given initial model parameter values are pre-set arbitrary defaults. In this example, the user accepts these values using the <RETURN> key.

2. Establish a model component after data input:

```
sherpa> ERASE ALL
sherpa> DATA example.pha
sherpa> POW[modelc]
modelc.gamma parameter value [0]
modelc.ref parameter value [1]
modelc.ampl parameter value [0.0010453]
sherpa> SHOW modelc
powlaw1d[modelc]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1          0         94
 3   ampl thawed     0.0010     0.0000     0.0021
```

In this example, the `ERASE ALL` command is used to remove the previous model setup. Next, data are input, with the command `DATA example.pha`. This is then followed by the establishment of the model component, with the command `POW[modelc]`. Notice that the given initial model parameter values and ranges are estimates based on the filtered data. Compare this example to example 1.

3. Establish a model component prior to data input; obtain parameter estimates using `GUESS`:

```
sherpa> ERASE ALL
sherpa> POW[modelc]
modelc.gamma parameter value [0]
modelc.ref parameter value [1]
modelc.ampl parameter value [1]
sherpa> DATA example.pha
sherpa> SHOW modelc
powlaw1d[modelc]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1-3.4028e+38 3.4028e+38
 3   ampl thawed          1     0.0000 3.4028e+38
sherpa> GUESS modelc
sherpa> SHOW modelc
powlaw1d[modelc]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  gamma thawed          0        -10         10
 2    ref frozen          1          0         94
 3   ampl thawed     0.0010     0.0000     0.0021
```

In this example, the `ERASE ALL` command is used to remove the previous input data and model component setups. Next, a model component is established prior to input of data (with the command `POW[modelc]`), and so the given initial parameter values are default values. The `GUESS` command is then used to obtain parameter value and range estimates, based on the input dataset.

5.4.3 Setting Model Parameters and Ranges

The model components that have been established in the current *Sherpa* session, and their parameter information, may be listed with the command `SHOW MODELS`. Values for these established model component parameters may be set individually using one of the following command syntax options:

```
sherpa> [CREATE] <modelname>.{<paramname> | <#>} {= <value> | => <paramExpr>}
sherpa> [CREATE] <modelname>.{<paramname> | <#>}.VALUE = <value>
```

where:

Argument:	Description:
`<modelname>`	A name given to a model component by the user.
`{<paramname> | <#>}`	Specifies the parameter whose value is to be set.
`<value>`	A number value to which the parameter should be set.
`=> <paramExpr>`	A syntax for linking parameters (see Section 5.4.12).

Note that either `<paramname>` or `<#>` may specify the parameter whose value is to be set.

Note also that parameter values may be set interactively from a plot: Please see command `GETX` (Section 1.39) for information on interactively assigning an x-axis value to a model parameter, from a plot; see command `GETY` (Section 1.40) for information on interactively assigning a y-axis value.

In addition, the `minimum`, `maximum`, and initial `delta` for parameters may also be individually set, using one of the following applicable command syntax options:

```
sherpa> [CREATE] <modelname>.{<paramname> | <#>}.MIN = <value>
sherpa> [CREATE] <modelname>.{<paramname> | <#>}.MAX = <value>
sherpa> [CREATE] <modelname>.{<paramname> | <#>}.DELTA = <value>
```

Examples:

1. Establish a model component, and assign it a name; set a model parameter value:

   ```
   sherpa> ERASE ALL
   sherpa> PARAMPROMPT OFF
   Model parameter prompting is off
   sherpa> POLY[modela]
   sherpa> modela.c0 = 3.0
   ```

   The command (`modela.c0 = 3.0`) gives the value of 3.0 to parameter `c0`, of `modela`. Note that `modela` is the name given by the user to the *Sherpa* model component `POLYNOM1D`. The following commands are each equivalent to the final command:

   ```
   sherpa> modela.c0.VALUE = 3.0
   sherpa> modela.1 = 3.0
   sherpa> modela.1.VALUE = 3.0
   ```

2. Establish a model component, and assign it a name; set a model parameter value, and the parameter range limits:

```
sherpa> POW[modelc]
sherpa> modelc.3 = 10.0
sherpa> modelc.3.MIN = 1.0
sherpa> modelc.3.MAX = 100.0
```

The second command, `modelc.3 = 10.0`, gives the value of 10.0 to parameter number 3, of `modelc`. The third and fourth commands set the minimum and maximum values for parameter number 3 of this model component.

3. Set initial model parameter step size `delta`:

```
sherpa> modelc.3.DELTA = 0.05
```

This command sets the parameter's initial step size, `delta`, to a value of 0.05.

4. Establish two different model components, and assign them names; link one model parameter to another:

```
sherpa> ERASE ALL
sherpa> POLY[modela]
sherpa> GAUSS[modelb]
sherpa> modelb.ampl => 0.5*modela.c0
```

The last command in this series uses a model parameter expression (see Section 5.4.12), to link the `ampl` parameter of `modelb` to 0.5 multiplied by the `c0` parameter of `modela`.

5.4.4 Setting Model Parameters and Ranges Simultaneously

Alternatively, model parameters and ranges may all be set at one time, using a single command:

```
sherpa> [CREATE] <modelname>.{<paramname> | <#>} = <param>
```

where again `<modelname>` is the name that has been given to a model component by the user. Note that either `<paramname>` or `<#>` may specify the parameter whose value is to be set. The command argument `<param>` contains the parameter assignments

`<value>:<min>:<max>:<delta>`, where:

Argument:	**Description:**
`<value>`	The value for the model parameter.
`<min>`	The minimum for that parameter range.
`<max>`	The maximum for that parameter range.
`<delta>`	Specifies initial parameter step size.

Examples:

1. Establish a model component, and assign it a name; set a model parameter value, and the parameter range limits:

```
sherpa> POW[modelc]
sherpa> modelc.3 = 10.0:1.0:100.0
```

The command `modelc.3 = 10.0:1.0:100.0`, gives the value of 10.0 to parameter number 3, sets the minimum to 1.0 for this parameter, and sets the maximum to 100.0 for this parameter, of `modelc` (where `modelc` is the name given by the user to the *Sherpa* model component `POWLAW1D`, with the second command). The following command is equivalent to the third command:

```
sherpa> modelc.ampl = 10.0:1.0:100.0
```

Compare the syntax of this example to Section 5.4.3, Example 2.

2. Set a model parameter value, the parameter range limits, and set the initial parameter step size `delta`:

```
sherpa> modelc.3 = 10.0:1.0:100.0:0.05
```

This command gives the value of 10.0 to `modelc` parameter number 3, sets the minimum to 1.0 for this parameter, sets the maximum to 100.0 for this parameter, and sets the parameter's initial step size, `delta`, to a value of 0.05. Compare the syntax of this example to Section 5.4.3, Example 3.

5.4.5 Establishing Model Components and Setting Parameters Simultaneously

To be especially efficient, one may establish and assign a name to a model component, as well as set model parameters and their ranges, all at one time using a single command:

```
sherpa> [CREATE] <sherpa_modelname>[<modelname>](<param> [,<param>,<param>, ...])
```

where again <**sherpa_modelname**> is the *Sherpa* default model name and <**modelname**> is the name being given to the model component by the user. Note that <**modelname**> *must be enclosed in brackets*, [ ]. The sequential order of the comma-separated <**param**> arguments indicates to which model parameter the values will be applied. That is, the first <**param**> argument will apply to the model's first parameter, the second <**param**> to the model's second parameter, *etc.*

As in the above Subsection, the command argument <**param**> contains the parameter assignments <**value**>:<**min**>:<**max**>:<**delta**>, where:

Argument:	**Description:**
<**value**>	The value for the model parameter.
<**min**>	The minimum for that parameter range.
<**max**>	The maximum for that parameter range.
<**delta**>	Specifies initial parameter step size.

The colon separated list for <**param**> need not include all of the elements, but it does need to maintain the proper sequential parameter order.

Examples:

1. Establish a model component, assign it a name, and set parameter values and ranges:

   ```
   sherpa> ERASE ALL
   sherpa> POLY[modela](3.0:1.0:4.0)
   ```

   Assigns the name **modela** to the *Sherpa* model component POLYNOM1D. Also gives the value of 3.0 to the model's first parameter (in this case parameter **c0**), sets the minimum to 1.0 for this parameter, and sets the maximum to 4.0 for this parameter.

2. Establish a model component, assign it a name, and set parameter values:

   ```
   sherpa> POW[modelc](1, 5, 10)
   ```

   Assigns the name **modelc** to the *Sherpa* model component POWLAW1D. Also gives the value of 1.0 to the model's first parameter (in this case parameter **gamma**), gives the value of 5.0 to the model's second parameter (**ref**), and gives the value of 10.0 to the model's third parameter (**ampl**).

3. Establish a model component, assign it a name, and set parameter values:

   ```
   sherpa> POW[modelc2](,,10.0:1.0:100.0:0.05)
   ```

   Assigns the name **modelc2** to the *Sherpa* model component POWLAW1D. Also gives the value of 10.0 to the model's third parameter (in this case parameter **ampl**), sets the minimum to 1.0 for this parameter, sets the maximum to 100.0 for this parameter, and set the initial step size to 0.05 for this parameter.

5.4.6 Establishing Model Components, Setting Parameters, & Defining a Source Model Simultaneously

Model component names may be assigned, model parameters and ranges may be set, and a source or background model may be defined, using a single command:

sherpa> {SOURCE | BACKGROUND} [#] = <**sherpa_modelname**>[<**modelname**>](<**param**>
[,<**param**>,<**param**>, ...])

where again <**sherpa_modelname**> is the *Sherpa* default model name and <**modelname**> is the name being given to the model component by the user. Note that <**modelname**> *must be enclosed in brackets*, [ ]. The sequential order of the comma-separated <**param**> arguments indicates to which model parameter the values will be applied. That is, the first <**param**> argument will apply to the model's first parameter, the second <**param**> to the model's second parameter, *etc.*

The
command argument <**param**> contains the parameter assignments <**value**>:<**min**>:<**max**>:<**delta**>, where:

Argument:	**Description:**
<**value**>	The value for the model parameter.
<**min**>	The minimum for that parameter range.
<**max**>	The maximum for that parameter range.
<**delta**>	Specifies initial parameter step size.

The colon separated list for <param> need not include all of the elements, but it does need to maintain the proper sequential parameter order.

Please also see the `SOURCE` command (Section 1.77) and the `BACKGROUND` command (Section 1.9) for more information and examples.

Examples:

1. Establish a model component, assign it a name, define the parameters, and define it as the source model to be used for fitting:

```
sherpa> ERASE ALL
sherpa> SOURCE = POLY[modela](3.0:1.0:4.0)
```

   With this single command, the name `modela` is assigned to the *Sherpa* model component `POLYNOM1D`, the value of 3.0 is given to the model's first parameter (in this case parameter `c0`), the minimum of 1.0 is set for this parameter, the maximum of 4.0 is set for this parameter, and then this model is defined as the source model to be used for fitting dataset number 1.

2. Establish multiple model components, assign them names, and use them in a source model expression definition:

```
sherpa> SOURCE 2 = GAUSS[modelb](3:2.5:4.403, 1:-10:10, 1:-3.5:3.5) + POW[modelc]
```

   With this single command, the following is performed: the name `modelb` is assigned to the *Sherpa* model component `GAUSS1D`; various parameter values and ranges are set for the parameters of `modelb`; the name `modelc` is assigned to the *Sherpa* model component `POWLAW1D`; a model expression (see Section 5.4.7), which is the sum of these two models, is defined as the source model to be used for fitting dataset number 2.

5.4.7 Creating Model Expressions

Model components can be used to create model expressions, in conjunction with the `SOURCE` (see Section 1.77) and `BACKGROUND` (see Section 1.9) commands (which then are used to fit the data), or with model stacks (see Section 5.4.8):

```
sherpa> {SOURCE | BACKGROUND | <model_stackname>} = <modelExpr>
```

Where the model expression, <modelExpr>, may be composed of one, or more (combined algebraicly), of the following elements:

Model Expression Element:	**Description:**
<sherpa_modelname>	A *Sherpa* default model name.
<sherpa_modelname>[modelname]	A name that is being given to a *Sherpa* model component (<modelname> *must be enclosed in brackets*, [ ]).
<modelname>	A name that has been given to a *Sherpa* model component.
<nested_model>	A nested model (see Section 5.4.9).
<jointmode_model>	A joint-mode model (see Section 5.4.10).

A model expression may be an algebraic combination of these elements that may include numerical values and the following operators: `+ - * / ( )` $<$ $>$ $\{\}$.

Note that by default, if a model expression includes a model component that has not previously been established, *Sherpa* will prompt the user for the initial model parameter values for that model component. Parameter prompting can be turned off using the `PARAMPROMPT OFF` command (see Section 1.58).

Examples:

1. Create a simple source model expression containing one model component:

   ```
   sherpa> PARAMPROMPT OFF
   Model parameter prompting is off
   sherpa> POLY[modela]
   sherpa> SOURCE 1 = modela
   ```

   The last command in this series assigns a model expression to the source model for dataset number 1. This model expression contains one model component, `modela` (which is an arbitrary name that the user has given to the *Sherpa* model component `POLYNOM1D`).

2. Create a source model expression containing multiple model components:

   ```
   sherpa> ATTEN[modelm]
   sherpa> BBODY[modelh]
   sherpa> SOURCE 1 = modelm * modelh
   ```

   This example establishes two model components, `modelm` and `modelh` (which are arbitrary names that the user has given to the *Sherpa* model components `ATTEN` and `BBODY`, respectively). The last command assigns the model expression `modelm * modelh` to the source model for dataset number 1. The following commands each assign various other model expressions to source models for other datasets:

   ```
   sherpa> SOURCE 2 = 10*(modelm + modelh)
   sherpa> SOURCE 3 = (modelm - modelh)/2
   sherpa> SOURCE 4 = 0.5*modelm + 0.7*modelh
   ```

3. Create a source model expression and establish model components simultaneously:

   ```
   sherpa> ERASE ALL
   sherpa> SOURCE 1 = ATTEN[modelm] * BBODY[modelh]
   ```

   In this example the user assigns the model expression `ATTEN[modelm] * BBODY[modelh]` to the source model for dataset number 1. Note that this single command is equivalent to the first series of commands in Example 2, above.

4. Create a background model expression:

   ```
   sherpa> ERASE ALL
   sherpa> PARAMPROMPT ON
   Model parameter prompting is on
   sherpa> BACKGROUND = (POW[modelc])/2
   modelc.gamma parameter value [0]
   modelc.ref parameter value [1]
   modelc.ampl parameter value [1]
   ```

This command assigns the model expression (POW[modelc])/2, to the background model (see Section 1.9) for dataset number 1. In this example, via parameter prompting, the user accepted the given initial values for all of the parameters, using the <RETURN> key.

5. Create a source model expression that includes a nested model:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> shape = GAUSS
sherpa> SIN[modeli]
sherpa> COS[modelj]
sherpa> SRC = modeli{shape} + modelj
```

The second command, shape = GAUSS, creates a model stack (see Section 5.4.8). The commands SIN[modeli] and COS[modelj] establish and assign names to the *Sherpa* model components SIN and COS, respectively. The final command assigns the model expression modeli{shape} + modelj, to the source model for dataset number 1. In this example, the model expression is an algebraic combination of a nested model (see Section 5.4.9), modeli{shape}, and a model component, modelj. Note that modeli will be evaluated on the output of the model expression shape (*i.e.*: SRC = SIN(GAUSS(x)) + COS(x)).

6. Create a source model expression involving joint mode 2-D models:

```
sherpa> DATA example_img.fits FITSIMAGE
sherpa> POLY[modelaForAxis1]
sherpa> GAUSS[modelbForAxis2]
sherpa> SRC = modelaForAxis1{x1}*modelbForAxis2{x2}
```

The command POLY[modelaForAxis1] assigns the name modelaForAxis1 to the *Sherpa* model component POLYNOM1D; likewise the command GAUSS[modelbForAxis2] assigns the name modelbForAxis2 to the *Sherpa* model component GAUSS1D. The final command assigns the model expression modelaForAxis1{x1}*modelbForAxis2{x2}, to the source model for dataset number 1. This model expression is an algebraic combination of joint-mode models (see Section 5.4.10).

5.4.8 Creating Model Stacks

A model stack assigns an arbitrary name to a model expression (see Section 5.4.7), for subsequent use in parameter expressions (see Section 5.4.12) and/or nested models (see Section 5.4.9).

sherpa> <model_stackname> = <modelExpr>

where <model_stackname> is the name being assigned to the model expression.

See Section 5.4.7 Example 5, Section 5.4.9, and Section 5.4.12 Example 3 for further examples.

Examples:

1. Create a model stack for use in a parameter expression:

```
sherpa> ERASE ALL
sherpa> GAUSS[modelb]
sherpa> SRC = modelb
sherpa> PositionVariation = POLY
sherpa> modelb.pos => PositionVariation
```

The command `GAUSS[modelb]`, assigns the name `modelb` to the *Sherpa* model component `GAUSS1D`. The next command defines this model component as the source model to be used for fitting. The third command, `PositionVariation = POLY`, assigns a model stack name to the *Sherpa* default model `POLYNOM1D`. The final command creates a parameter expression that links the parameter `pos` of `modelb` to the model component `POLY` (via the model stack name `PositionVariation`). Note that the creation of a model stack is necessary since the following syntax is currently *not allowed*:
`modelb.pos => POLY`.

2. Create multiple model stacks for use in parameter expressions:

```
sherpa> ERASE ALL
sherpa> SRC = GAUSS[modelb]
sherpa> PositionVariation = POLY[modela]
sherpa> fwhmVariation = POLY[modelaa]
sherpa> modelb.pos => PositionVariation
sherpa> modelb.fwhm => fwhmVariation
```

This example creates two different model stacks, and assigns the names `PositionVariation` and `fwhmVariation` to the model stacks. These model stack names are then used in parameter expressions, to link different parameters of `modelb` to different model components: the command `modelb.fwhm => fwhmVariation`, for example, links the parameter `fwhm` of `modelb` to the model component `modelaa` (via the model stack name `fwhmVariation`).

3. Create a model stack for use in a nested model:

```
sherpa> SHLOGE[modelk]
sherpa> independent = SIN[modeli] + COS[modelj]
sherpa> SOURCE = modelk{independent}
```

The command `SHLOGE[modelk]` assigns the name `modelk` to the *Sherpa* model component `SHLOGE`. Next, the user assigns a model stack name (`independent`) to the model expression `SIN[modeli] + COS[modelj]`. With the final command, the user then assigns the model expression `modelkindependent`, to the source model for dataset number 1. This source model expression is a nested model (see Section 5.4.9), which utilizes the model stack. Note that the model stack definition is necessary since the following syntax is *not allowed*:
`SOURCE = modelk{SIN[modeli] + COS[modelj]}`.

5.4.9 Creating Nested Models

Nested models define the function argument, on which a model expression is evaluated, to be another model expression.

A nested model, <`nested_model`>, appears as an element of a model expression, <`modelExpr`> (see Section 5.4.7), and is constructed using one of the following syntax options:

<**sherpa_modelname**>**{**<**model_stackname**>**}**
<**sherpa_modelname**>[<**modelname**>]**{**<**model_stackname**>**}**
<**modelname**>**{**<**model_stackname**>**}**

where <**model_stackname**> *must be enclosed in curly braces*, { }, and where the model stack (see Section 5.4.8) defines the model expression (see Section 5.4.7) for the function argument.

See Section 5.4.7 Example 5, and Section 3, Example 3, for further examples.

Examples:

1. Create a nested model that defines one model component as the argument for another:

   ```
   sherpa> ERASE ALL
   sherpa> oscillation = SIN
   sherpa> SRC = SHLOGE{oscillation}
   ```

   The user first assigns a model stack name (see Section 5.4.8) to the *Sherpa* default model `SIN`. The final command establishes the nested model, `SHLOGE{oscillation}`, and assigns it to the source model (see Section 1.77) for dataset number 1. This nested model defines the function argument `SIN`, over which the model component `SHLOGE` is to be evaluated. Note that the following syntax is *not allowed*: `SRC = SHLOGE{SIN}`.

2. Create two layers of nested models:

   ```
   sherpa> stackA = POLY
   sherpa> stackB = GAUSS{stackA}
   sherpa> SIN[modeli]
   sherpa> SOURCE = modeli{stackB}
   ```

   The command `stackA = POLY` creates the first model stack. The next command creates a second model stack, using the nested model construction, `GAUSS{stackA}`, where the function argument on which the model component `GAUSS` is evaluated is the first model stack. Finally, the source model expression is defined as the nested model `modeli{stackB}` (*i.e.* `modeli` evaluated upon the second model stack).

5.4.10 Creating Joint-Mode Models

Joint-mode models define the function argument, on which a model expression is evaluated, to be a particular data axis.

A joint-mode model, <**jointmode_model**>, may appear as an element of a model expression, <**modelExpr**> (see Section 5.4.7), and is constructed using one of the following syntax options:

<**sherpa_modelname**>**{**<**axis**>**}**
<**sherpa_modelname**>[<**modelname**>]**{**<**axis**>**}**
<**modelname**>**{**<**axis**>**}**

where <**axis**> *must be enclosed in curly braces*, { }, and where <**axis**> is a data column name. Note that <**axis**> defines the particular data axis on which the model expression is to be evaluated.

See Section 5.4.7 Example 6 for an additional example.

Examples:

1. Create 2-D joint-mode models that define data axes as the arguments for model components:

```
sherpa> DATA example_img.fits FITSIMAGE
sherpa> LORENTZ[SpatialModelAxis0](98:5:200, 70:50:90, 1:1:200)
sherpa> POWLAW1D[SpecModelAxis1]
sherpa> SRC 1 = SpatialModelAxis0{x1}*SpecModelAxis1{x2}
```

The command `LORENTZ[SpatialModelAxis0](98:5:200, 70:50:90, 1:1:200)` assigns the name `SpatialModelAxis0` to the *Sherpa* model component `LORENTZ1D`, and assigns various parameters values and ranges. Similarly, a second model component is established and assigned a name. The final command assigns the model expression `SpatialModelAxis0{x1}*SpecModelAxis1{x2}`, to the source model for dataset number 1. This source model expression is an algebraic combination of joint-mode models, where these joint-mode models each define a specific data axis as the argument for a model component. So with this source model expression, the joint-mode model `SpatialModelAxis0{x1}` will fit model `SpatialModelAxis0` to Axis 0 (`x1`) of the data, and the joint-mode model `SpecModelAxis1{x2}` will jointly fit model `SpecModelAxis1` to Axis 1 (`x2`) of the data, *etc.*

5.4.11 Freezing & Thawing Model Parameters

Model parameters may be frozen or thawed using one of the following equivalent commands:

```
sherpa> <modelname>.<paramname>.TYPE = {FREEZE | THAW}
sherpa> <modelname>.<#>.TYPE = {FREEZE | THAW}
sherpa> {FREEZE | THAW} <modelname>.{<paramname> | <#>} \
[ <modelname>.{<paramname> | <#>} ...]
```

In addition, all parameters of a model component may be frozen or thawed at once using:

```
sherpa> {FREEZE | THAW} <modelname>
```

And, all parameters of a source model's components may be frozen or thawed at once using:

```
sherpa> {FREEZE | THAW} SOURCE [#]
```

where `#` may specify the number of the dataset (default dataset number is 1).

With parameter prompting on, a model parameter may be frozen by entering `[<value>:<min>:<max>],-1` at the model parameter value prompt, where the `-1` sets the parameter to be frozen. (see Example 2, below).

See the `FREEZE` command (Section 1.37) and the `THAW` command (Section 1.83) for more information and further examples.

Examples:

1. Freeze a model parameter:

```
sherpa> PARAMPROMPT OFF
sherpa> GAUSS[modelb]
sherpa> modelb.ampl.TYPE = FREEZE
```

   This command freezes the parameter `ampl` of `modelb`. The following commands are each equivalent:

```
sherpa> modelb.3.TYPE = FREEZE
sherpa> FREEZE modelb.ampl
sherpa> FREEZE modelb.3
```

2. Freeze model parameters:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> GAUSS[modelb]
modelb.fwhm parameter value [10] 2,-1
modelb.pos parameter value [0] ,-1
modelb.ampl parameter value [1]
sherpa> SHOW modelb
gauss1d[modelb]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen          2 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
```

   In this example, parameters are frozen by entering <`value`>`, -1` at the model parameter value prompt. Note that `, -1` at the model parameter value prompt accepts the given initial parameter value and freezes that parameter.

3. Freeze all model parameters at once; thaw all source parameters at once:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> DATA data/example.pha
sherpa> GAUSS[modelc]
sherpa> SOURCE = modelb + modelc
sherpa> FREEZE modelc
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen          2 1.1755e-38 3.4028e+38
 2    pos frozen          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
```

```
 1   fwhm frozen     0.7113     0.0071    71.1283
 2    pos frozen     0.9442     0.0276    14.5494
 3   ampl frozen     0.0001 1.0564e-06     0.0106
sherpa> THAW SOURCE
sherpa> SHOW SOURCE
(modelb + modelc)
gauss1d[modelb]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed          2 1.1755e-38 3.4028e+38
 2    pos thawed          0-3.4028e+38 3.4028e+38
 3   ampl thawed          1-3.4028e+38 3.4028e+38
gauss1d[modelc]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed     0.7113     0.0071    71.1283
 2    pos thawed     0.9442     0.0276    14.5494
 3   ampl thawed     0.0001 1.0564e-06     0.0106
```

Note that the command `FREEZE modelc` freezes all parameters of the source model component `modelc`, while `THAW SOURCE` thaws all parameters of both source model components.

5.4.12 Linking Model Parameters

Parameter expressions are used to link model component parameters:

`sherpa> [CREATE] <modelname>.{<paramname> | <#>} => <paramExpr>`

where again `<modelname>` is the name that has been given to a model component by the user. Note that either `<paramname>` or `<#>` may specify a parameter. The parameter expression, `<paramExpr>`, to which the parameter is being linked, may be composed of one, or more (combined algebraicly), of the following elements:

Element:	**Description:**
`<modelname>.{<paramname>	<#>}`	A model component parameter.
`<model_stackname>`	A model expression, specified via a model stack name.

A parameter expression may be an algebraic combination of these elements that may include numerical values and the following operators: `+ - * / ( ) < > {}`.

The command `UNLINK` (see Section 1.87) may be used to remove a link between model parameters:

`sherpa> UNLINK <modelname>.{<paramname> | <#>}`

Examples:

1. Link one model parameter to another:

```
sherpa> ERASE ALL
sherpa> PARAMPROMPT ON
Model parameter prompting is on
sherpa> GAUSS[modelb]
modelb.fwhm parameter value [10]
modelb.pos parameter value [0]
modelb.ampl parameter value [1]
sherpa> GAUSS[modelf]
modelf.fwhm parameter value [10]
modelf.pos parameter value [0]
modelf.ampl parameter value [1]
sherpa> modelf.ampl => 0.5*modelb.ampl
```

The last command in this series uses a model parameter expression, to link the `ampl` parameter of `modelf` to 0.5 multiplied by the `ampl` parameter of `modelb`. That is, the amplitudes of two Gaussian models are linked, where one is half that of the other. *Note that model parameter expressions cannot be created within the model parameter prompting.* Note that the link may be removed as follows:

```
sherpa> UNLINK modelf.ampl
```

2. Link one model parameter to another:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> POW[modelc]
sherpa> POW[modele]
sherpa> modelc.1 => modele.1
sherpa> modelc.3 => 2*modele.3 - modele.3
```

The next-to last command in this series links the first parameter (`gamma`) of `modelc` to the first parameter of `modele`. The last command in this series links the third parameter (`ampl`) of `modelc` to the parameter expression: 2 multiplied by the third parameter of `modele`, minus the third parameter of `modele`. Note that the links may be removed as follows:

```
sherpa> UNLINK modelc.1
sherpa> UNLINK modelc.3
```

3. Link a model parameter to a model expression:

```
sherpa> ERASE ALL
sherpa> SIN[modeli]
sherpa> SOURCE = modeli
sherpa> PeriodVariation = POLY[modela]
sherpa> modeli.period => PeriodVariation
```

The first command, `SIN[modeli]`, assigns the name `modeli` to the *Sherpa* model component `SIN`. The second command defines this model component as the source model to be used for fitting. The third command, `PeriodVariation = POLY[modela]`, creates a model stack (see Section 5.4.8). The final command links the parameter `period` of `modeli` to the parameter expression `PeriodVariation` (which is composed of the model component `modela`). Note that the following parameter expression syntax is currently *not allowed*:
`modeli.period => POLY[modela]`.

5.4.13 Controlling the Integration of Model Components

The user can turn on and off the integration of non-*XSPEC*, non-instrument model components:

`sherpa> <modelname> INTEGRATE <ON|OFF>`

Changing the integration status of a model component will only have an affect if binned data are being fit (*e.g.* PHA or histogram data).

The default integration status for each model is provided in Table 5.9, while the `SHOW` command (Section 1.76) can be used to determine whether a model component is currently being integrated.

The user must be careful not to define an unacceptable mixture of additive/integrated and multiplicative/non-integrated model components, such as by adding two model components with different integration statuses. *Sherpa* will issue a warning when it detects such an unacceptable mixture.

Example:

1. Turn off integration for a power-law model component:

```
sherpa> POWLAW1D[p]
sherpa> SHOW p
powlaw1d[p]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
  1  gamma thawed         0        -10         10
  2    ref frozen         1-3.4028e+38 3.4028e+38
  3   ampl thawed         1      1e-20 3.4028e+38
sherpa> p INTEGRATE OFF
sherpa> SHOW p
powlaw1d[p]  (integrate: off)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
  1  gamma thawed         0        -10         10
  2    ref frozen         1-3.4028e+38 3.4028e+38
  3   ampl thawed         1      1e-20 3.4028e+38
```

5.5 Summary of Available Models

<sherpa_modelname>	**Description**	**Integration**
`ATTEN`	Attenuation by ISM.	OFF
`BBODY | BB`	Blackbody as a function of energy.	ON
`BBODYFREQ`	Blackbody as a function of frequency.	ON
`BETA1D`	1-D surface brightness beta-model.	OFF
`BETA2D | LORPOW2D`	2-D Lorentzian with varying power law.	ON
`BOX1D`	1-D box function.	OFF
`BOX2D`	2-D box function.	OFF
`BPL1D`	Broken power law function.	ON

<sherpa_modelname>	**Description**	**Integration**
CONST1D | CONST	1-D constant amplitude model.	ON
CONST2D	2-D constant amplitude model.	ON
COS	Cosine function.	OFF
DELTA1D	1-D delta function.	ON
DELTA2D	2-D delta function.	ON
DEVAUCOULEURS	DeVaucouleurs profile.	ON
DERED	Dereddening function.	OFF
EDGE	Photoabsorption edge model.	OFF
ERF	A 1-D error function.	OFF
ERFC	A 1-D complementary error function.	OFF
FARF1D | FARF	A 1-D file-based ancillary response model	N/A
FARF2D | FEXPMAP | FEXPMAP2D	A 2-D file-based ancillary response model	N/A
FPSF1D	A 1-D file-based PSF instrument model	N/A
FPSF2D | FPSF | PSFFROMFILE	A 2-D file-based PSF instrument model	N/A
FRMF1D | FRMF	A 1-D file-based response matrix model	N/A
GAUSS1D | GAUSS	1-D unnormalized Gaussian function.	ON
GAUSS2D	2-D unnormalized Gaussian function.	ON
GRIDMODEL	N-D user-specified amplitude model.	OFF
HIGHPASS | STEPHI1D	1-D step function.	OFF
HUBBLE | REYNOLDS	Hubble-Reynolds profile.	ON
JDPILEUP	John Davis (MIT) pileup model.	OFF
KERNEL	A 1-D convolution model.	N/A
LINEBROAD	Line broadening profile.	ON
LORENTZ1D | LORENTZ	1-D normalized Lorentzian function.	ON
LORENTZ2D	2-D unnormalized Lorentzian function.	ON
LORPOW2D | BETA2D	2-D Lorentzian with varying power law.	ON
LOWPASS | STEPLO1D	1-D step function.	OFF
NBETA	1-D normalized beta function.	ON
NGAUSS1D	1-D normalized Gaussian function.	ON
PILEUP	1-D pileup model function.	N/A
POISSON	Poisson function.	OFF
POLYNOM1D | POLY	1-D polynomial function.	ON
POLYNOM2D	2-D polynomial function.	ON
POWLAW1D	1-D power law.	ON
PTSRC1D	A 1-D file-based point-source fitting model	OFF
PTSRC2D | PTSRC	A 2-D file-based point-source fitting model	OFF
REYNOLDS | HUBBLE	Hubble-Reynolds profiles.	ON
RSP1D | RSP	A 1-D instrument model.	N/A
RSP2D	A 2-D instrument model.	N/A
SCHECHTER	Schechter function.	OFF
SHEXP10	Exponential function, base 10.	OFF
SHEXP	Exponential function.	OFF
SHLOG10	Logarithm function, base 10.	OFF
SHLOGE | SHLOG	Natural logarithm function.	OFF
SIN	Sine function.	OFF
SQRT	Square root function.	OFF
STEPHI1D | HIGHPASS	1-D step function.	OFF
STEPLO1D | LOWPASS	1-D step function.	OFF
TAN	Tangent function.	OFF
TPSF1D	A 1-D TCD-model-based PSF instrument model	N/A

<sherpa_modelname>	Description	Integration
`TPSF2D | TPSF |PSFFROMTCD | PSF`	A 2-D TCD-model-based PSF instrument model	N/A
`USERMODEL`	User implemented model.	OFF
`XS<xspecname>`	An *XSPEC*model function.	N/A

Table 5.9: Summary of Models

The last column in Table 5.9 indicates whether the model is integrated by default when fit to binned data: `ON` indicates that the model is integrated by default. In section 5.4.13, we describe the `INTEGRATE <ON|OFF>` command, which allows the user to change the integration status of a non-*XSPEC*, non-instrument model components.

In the remainder of this chapter, we describe each of the models listed above, in alphabetical order. Each model section includes a description of the model, and a table listing of the model parameters. The listing of model parameters includes the *Sherpa* parameter number, and the corresponding *Sherpa* parameter name (`<paramname>`).

5.6 ATTEN

Attenuation by ISM. Integration OFF.

This model calculates the transmission of the interstellar medium using the description of the ISM absorption of Rumph, Bowyer, & Vennes 1994, AJ 107, 2108. It includes neutral He autoionization features. Between 1.2398 and 43.655 Angstroms (*i.e.* in the 0.28-10 keV range) the model also accounts for metals as described in Morrison & MacCammon 1983, ApJ 270, 119.

The code uses the best available photoionization cross-sections to date from the atomic data literature and combines them in an arbitrary mixture of the three ionic species: HI, HeI, and HeII.

The model assumes that the data are expressed in Angstroms.

This model provided courtesy of Pat Jelinsky.

Number	Name	Description
1	`hcol`	N(HI) column (atoms cm^{-2})
2	`heiRatio`	N(HeI)/N(HI)
3	`heiiRatio`	N(HeII)/N(HI)

Table 5.10: ATTEN Parameters

5.7 BBODY — BB

Blackbody as a function of energy. Integration ON.

Blackbody emission calculated as a function of energy using the expression:

$$f(E) = A\frac{E^2}{\exp(\frac{E}{kT}) - 1} \tag{5.2}$$

where E is the photon energy, and kT is the blackbody temperature (expressed in the same units as the photon energy). The amplitude A is related to the ratio of source radius to distance:

$$A = \frac{2\pi}{c^2h^3}\left(\frac{R}{d}\right)^2 \;=\; 9.884{\times}10^{31}\left(\frac{R}{d}\right)^2, \tag{5.3}$$

with Planck's constant h specified in keV-s and the speed of light c specified in cm/s, and with R and d representing the radius of, and distance to, the source respectively. If $E/kT < 10^-4, f(E) = AEkT$, while if $E/kT > 60, f(E) = 0$.

Number	Name	Description
1	`space`	0: energy | 1: wavelength
2	`kT`	temperature kT (in energy units)
3	`ampl`	amplitude A

Table 5.11: BBODY Parameters

5.8 BBODYFREQ

Blackbody as a function of frequency. Integration ON.

Blackbody emission calculated as a function of frequency ν using Wien's law ($h\nu \gg kT$):

$$f(\nu) = A\frac{2h\nu^3}{c^2}\exp(-\frac{h\nu}{kT}) \tag{5.4}$$

where A is the amplitude, T is the temperature in Kelvin, and $h, c,$ and k are Planck's constant, the speed of light, and Boltzmann's constant, respectively.

Number	Name	Description
1	`T`	temperature T (K)
2	`ampl`	amplitude A

Table 5.12: BBODYFREQ Parameters

5.9 BETA1D

1-D surface brightness beta-model. Integration OFF

1-D surface brightness beta-model:

$$f(x) = A\left(1 + \left[\frac{x - x_o}{r_o}\right]^2\right)^{-3\beta+\frac{1}{2}} \tag{5.5}$$

Number	Name	Description
1	`r_o`	core radius r_o
2	`beta`	β index
3	`xpos`	offset x_o from $x = 0$
4	`ampl`	amplitude A at $x = x_o$

Table 5.13: BETA1D Parameters

5.10 BETA2D — LORPOW2D

2-D Lorentzian with varying power law. Integration OFF. The `LORPOW2D` model is equivalent.

A 2-D Lorentz model with a varying power law, also known as a Beta model:

$$f(x,y) = f(r) \;=\; \frac{A}{(1 + (\frac{r}{r_0})^2)^{\alpha}} \tag{5.6}$$

where

$$r(x,y) = \frac{\sqrt{x_{\rm new}^2(1-\epsilon)^2 + y_{\rm new}^2}}{1-\epsilon} \tag{5.7}$$

$$x_{\rm new} = (x - x_{\rm o})\cos\theta + (y - y_{\rm o})\sin\theta \tag{5.8}$$

and

$$y_{\rm new} = (y - y_{\rm o})\cos\theta - (x - x_{\rm o})\sin\theta \tag{5.9}$$

Number	Name	Description
1	r0	core radius r_0
2	`xpos`	x mean position x_o

3	`ypos`	y mean position y_o
4	`ellip`	ellipticity ϵ
5	`theta`	angle of ellipticity θ
6	`ampl`	amplitude A at (x_o, y_o)
7	`alpha`	power law index α

Table 5.14: BETA2D Parameters

5.11 BOX1D

1-D box function. Integration OFF.

A 1-D box model:

$$f(x) = A \qquad \text{if } x_{\mathrm{low}} \leq x \leq x_{\mathrm{hi}} \tag{5.10}$$

$$f(x) = 0 \qquad \text{otherwise} \tag{5.11}$$

Number	Name	Description
1	`xlow`	low cut-off x_{low}
2	`xhi`	high cut-off x_{hi}
3	`ampl`	amplitude A

Table 5.15: BOX1D Parameters

5.12 BOX2D

2-D box function. Integration OFF.

A 2-D box model:

$$f(x, y) = A \tag{5.12}$$

if $x_{\mathrm{low}} \leq x \leq x_{\mathrm{hi}}$ and $y_{\mathrm{low}} \leq y \leq y_{\mathrm{hi}}$,

$$f(x, y) = 0 \qquad \text{otherwise} \tag{5.13}$$

Number	Name	Description
1	`xlow`	low x cut-off x_{low}

2	`xhi`	high x cut-off $x_{\rm hi}$
3	`ylow`	low y cut-off $y_{\rm low}$
4	`yhi`	high y cut-off $y_{\rm hi}$
5	`ampl`	amplitude A

Table 5.16: BOX2D Parameters

5.13 BPL1D

Broken power law function. Integration ON.

A broken power-law model:

$$f(x) = A \left(\frac{x}{x_{\rm ref}}\right)^{-\gamma_1} \tag{5.14}$$

if $x \leq x_b$, and

$$= A' \left(\frac{x}{x_{\rm ref}}\right)^{-\gamma_2} \tag{5.15}$$

otherwise, where

$$A' = A \left(\frac{x_b}{x_{\rm ref}}\right)^{\gamma_2 - \gamma_1} \tag{5.16}$$

Number	**Name**	**Description**
1	`gamma1`	first power law photon index γ_1
2	`gamma2`	second power law photon index γ_2
3	`eb`	break-point x_b
4	`ref`	normalization reference point x_{ref}
5	`ampl`	amplitude A

Table 5.17: BPL1D Parameters

5.14 CONST1D — CONST

1-D constant amplitude model. Integration ON.

A 1-D constant amplitude model:

$$f(x) = A \tag{5.17}$$

A is limited to being > 0. To model negative constant amplitudes, multiple by -1 in *Sherpa*:

```
sherpa> source = (-1)*const[cs]
```

Number	Name	Description
1	`c0`	amplitude A

Table 5.18: CONST1D Parameter:

5.15 CONST2D

2-D constant amplitude model. Integration ON.

A 2-D constant amplitude model:

$$f(x) = A \tag{5.18}$$

Number	Name	Description
1	`c0`	amplitude A

Table 5.19: CONST1D Parameter:

5.16 COS

Cosine function. Integration OFF.

A 1-D cosine model:

$$f(x) = A \cos\left(\frac{2\pi(x - x_{\text{off}})}{P}\right) \tag{5.19}$$

Number	Name	Description
1	`period`	period P, in units of x
2	`offset`	x offset x_{off}
3	`ampl`	amplitude A

Table 5.20: COS Parameters

5.17 DELTA1D

1-D delta function. Integration ON.

A 1-D delta-function model:

$$f(x) = A \tag{5.20}$$

if $x = x_o$, and

$$= 0 \tag{5.21}$$

otherwise.

with

$$\int_{-\infty}^{+\infty} dx f(x) \;=\; A\,. \tag{5.22}$$

Number	Name	Description
1	`pos`	position x_o
2	`ampl`	amplitude A

Table 5.21: DELTA1D Parameters

5.18 DELTA2D

2-D delta function. Integration ON.

A 2-D delta-function model:

$$f(x, y) = A \tag{5.23}$$

if $x = x_o$ and $y = y_o$,

and,

$$f(x, y) = 0 \tag{5.24}$$

otherwise.

with

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} dx dy f(x, y) \;=\; A \tag{5.25}$$

Number	Name	Description
1	`xpos`	x position x_o
2	`ypos`	y position y_o
3	`ampl`	amplitude A

Table 5.22: DELTA2D Parameters

5.19 DERED

Dereddening function. Integration OFF.

This dereddening model uses the analytic formula for the mean extension law described in Cardelli, Clayton, & Mathis 1989, ApJ 345, 245:

$$A(\lambda) = E(B - V)\,(aR_v + b) = 1.086\tau(\lambda) \tag{5.26}$$

where $\tau(\lambda)$ is the wavelength-dependent optical depth,

$$I(\lambda) = I(0)\exp\left(-\tau(\lambda)\right) \tag{5.27}$$

and a and b are computed using wavelength-dependent formulae which we will not reproduce here, for the wavelength range 1000 Å - 3.3 microns. The relationship between the color excess and the column density is

$$E(B - V) = \frac{N_{H\mathrm{gal}}(10^{20}\mathrm{cm}^{-2})}{58.0} \tag{5.28}$$

(Bohlin, Savage, & Drake 1978, ApJ 224, 132). The value of the ratio of total to selective extinction, R_v, is initially set to 3.1, the standard value for the diffuse ISM. The final model form is:

$$I(\lambda) = I(0)\exp\left[-\frac{N_{H\mathrm{gal}}\,[aR_v + b]}{58.0 \times 1.086}\right] \tag{5.29}$$

This model should only be used as a multiplicative model:

```
sherpa> powlaw1d[con1](1.,2588.6,0.1)
sherpa> dered[dr](3.1,0.1)
sherpa> source 1 = con1*dr
```

This model provided courtesy of Karl Forster.

Number	Name	Description

1	rv	total to selective extinction ratio R_v
2	nhgal	absorbing column density $N_{H\mathrm{gal}}$

Table 5.23: DERED Parameters

5.20 DEVAUCOULEURS

DeVaucouleurs profile. Integration OFF.

A 2-D de Vaucouleurs model:

$$f(x,y) = f(r) = A \exp\left[-7.67\left(\sqrt{\left[\frac{x_{\mathrm{maj}}}{r_{\mathrm{maj}}}\right]^2 + \left[\frac{x_{\mathrm{min}}}{r_{\mathrm{min}}}\right]^2}\right)^{0.25} - 1\right] \tag{5.30}$$

where

$$r_{\mathrm{maj}} = r_0 \tag{5.31}$$

$$r_{\mathrm{min}} = (1-\epsilon)r_0 \tag{5.32}$$

$$x_{\mathrm{maj}} = (x - x_{\mathrm{o}})\cos\theta + (y - y_{\mathrm{o}})\sin\theta \tag{5.33}$$

$$x_{\mathrm{min}} = -(x - x_{\mathrm{o}})\sin\theta + (y - y_{\mathrm{o}})\cos\theta \tag{5.34}$$

Number	**Name**	**Description**
1	r0	core radius r_0
2	xpos	x mean position $x_{/rmo}$
3	ypos	y mean position y_{o}
4	ellip	ellipticity ϵ
5	theta	angle of ellipticity θ
6	ampl	amplitude A at (x_o, y_o)

Table 5.24: DEVAUCOULEURS Parameters

5.21 EDGE

Photoabsorption edge model. Integration OFF.

A phenomenological photoabsorption edge model as a function of energy:

$$f'(x) = f(x) \tag{5.35}$$

if $x < E_b$, and

$$f(x) \exp\left[-A\left(\frac{x}{E_b}\right)^{-3}\right] \tag{5.36}$$

otherwise.

Or, as a function of wavelength:

$$f'(x) = f(x) \tag{5.37}$$

if $x > \lambda_b$, and

$$= f(x) \exp\left[-A\left(\frac{x}{\lambda_b}\right)^{3}\right] \tag{5.38}$$

otherwise.

Number	Name	Description
1	`space`	energy (0) or wavelength (1)
2	`thresh`	edge position E_b or λ_b
3	`abs`	absorption coefficient A

Table 5.25: EDGE Parameters

5.22 ERF

A 1-D error function. Integration OFF.

$$f(x) \;=\; A\,\mathrm{erf}\left(\frac{x - x_{\mathrm{off}}}{\sigma}\right) \tag{5.39}$$

where

$$\mathrm{erf}(y) \;=\; \frac{2}{\sqrt{\pi}} \int_0^y \exp\left[-t^2\right] dt \tag{5.40}$$

Number	Name	Description
1	`ampl`	amplitude A
2	`offset`	offset x_{off}
3	`sigma`	scaling factor σ

Table 5.26: ERF Parameters

erf is related to erfc, a complementary error function:

```
erfc(y) = 1 - erf(y)
```

See `ERFC` for information on that function.

5.23 ERFC

A 1-D complementary error function. Integration OFF.

$$f(x) \;=\; A\,\mathrm{erfc}\left(\frac{x - x_{\mathrm{off}}}{\sigma}\right) \tag{5.41}$$

where

$$\mathrm{erfc}(y) \;=\; \frac{2}{\sqrt{\pi}} \int_y^\infty \exp\left[-t^2\right] dt \tag{5.42}$$

Number	Name	Description
1	`ampl`	amplitude A
2	`offset`	offset x_{off}
3	`sigma`	scaling factor σ

Table 5.27: ERF Parameters

erfc is related to erf, a 1-D error function:

```
erfc(y) = 1 - erf(y)
```

See ERF for information on that function.

5.24 FARF1D — FARF

A 1-D file-based ancillary response model.

FARF is a file-based ARF model that represents an ancillary response vector (*e.g.*, the effective area as a function of energy/wavelength).

Note that in *Sherpa* analyses, the energy-space binning of the ARF (found typically in the ENERG_LO and ENERG_HI columns of the SPECRESP extension of the ARF file) does not have to match the analogous energy-space binning of a matching RMF. (This is also true if instead of energy, the ARF is defined as a function of wavelength.)

See the documentation on the INSTRUMENT command (Section 1.46). See also the RSP (Section 5.51) and FRMF (Section 5.28) instrument models.

Number	**Name**	**Description**
1	arf	ARF file name
2	norm	normalization

Table 5.28: FARF Parameters

5.25 FARF2D — FEXPMAP — FEXPMAP2D

A 2-D file-based ancillary response model. Alternate names include FEXPMAP and FEXPMAP2D.

FARF2D is a file-based model whose primary use is to represent 2-D exposure maps in arbitrary units (seconds, cm^2, *etc.*).

Note the following about exposure maps:

- The exposure map image must be of the same size as the data image, and it must have the same plate-scale (*i.e.*, bins in each image must have the same size in arc-seconds, *etc.*).
- If an exposure map and a point-spread function (PSF) are to be used in the same instrument stack, then the images containing both must have the same plate-scale.

See the documentation on the INSTRUMENT command (Section 1.46). See also the RSP2D (Section 5.51) instrument model.

Number	**Name**	**Description**
1	file	exposure map file name
2	norm	normalization

Table 5.29: FARF2D Parameters

Examples:

1. Input an exposure map for use with a dataset:

```
sherpa> DATA data/data.fits
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> FEXPMAP[emap]
sherpa> emap.file = "data/data_expmap.fits"
sherpa> INSTRUMENT = emap
```

In this example, the instrument model component `FEXPMAP` is established, the exposure map file is input using this model's file parameter, and the instrument model stack is set for dataset number 1.

The CIAO 2.3 equivalent of this command is:

```
sherpa> DATA data/data.fits
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GRIDMODEL[gm]
sherpa> gm.file = "data/data_expmap.fits"
sherpa> EXPMAP 1 = gm
```

2. Use a point-spread function file in addition to an exposure map:

```
sherpa> FEXPMAP[emap]
sherpa> emap.file = "data/data_expmap.fits"
sherpa> FPSF[psfunc]
sherpa> psfunc.file = "data/data_psf.fits"
sherpa> INSTRUMENT = emap*psfunc
```

This may also be done with the `RSP2D` command, Section 5.52,:

```
sherpa> RSP2D[r]
sherpa> r.empfile = "data/data_expmap.fits"
sherpa> r.psffile = "data/data_psf.fits"
sherpa> INSTRUMENT = r
```

5.26 FPSF1D

A 1-D file-based PSF instrument model.

`FPSF1D` is a file-based PSF model that represents the point-spread function, a redistribution function that maps photon locations to data bins.

This model cannot be used in a source model expression. See `PTSRC1D` model if you want to fit the data with the PSF model file.

ASCII and FITS formats are allowed for the input model file . The current implementation requires that the bin size of the data and the PSF model file are the same. The number of bins can be different.

Note that the input file is automatically renormalized to 1 upon entry. Renormalization is done by summing over all bins, regardless of the setting of xsize.

The input PSF array is used to convolve (fold) a given source model. This convolution can be performed using either Fast Fourier Transforms (FFTs, the default), or the sliding cell technique (see the parameter `fft`). If the axis length is N and the length of the kernel axis is $M (= 2 \times \texttt{xsize} + 1)$, then the computation time for the sliding cell goes as $N \times M$, *i.e.*, for large kernels the best choice is using the FFT. A rough rule-of-thumb for 1-D fits is to use the FFT if $M > 10$.

"xsize" parameter defines the number of bins extracted from the file to be used in the PSF model.

`xoff = 0`, the PSF (kernel) sub-array is extracted from the center of the original array contained in the input PSF `file`.

Note that the kernel centroid must always be at the center of the extracted sub-array! Otherwise, systematic shifts will occur in best-fit positions of point sources.

In *Sherpa* version 3.0.2, a new parameter is introduced: `radial`. If set to 1 the kernel array will be extended and its values reflected across the edge boundary. The resultant function will be symmetric. The default value is 0.

See the documentation on the `INSTRUMENT` command (Section 1.46).

Number	Name	Description
1	`file`	input file name (FITS or ASCII)
2	`xsize`	x-full-width of the subset region of kernel file to use in convolution
3	`xoff`	x-direction offset
4	`fft`	convolution type: 1 = FFT / 0 = sliding cell
5	`radial`	radial profile: 1 = yes / 0 = no

Table 5.30: FPSF1D Parameters

Examples:

1. The example below shows the initial parameter values for the `FPSF1D` model. Note that the default `xsize` value is set to 32 before the PSF data file is read in. After reading the data file the `xsize` is set to the size of the array in the model file.

```
sherpa> fpsf1d[psf0]
sherpa> show psf0
fpsf1d[psf0]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   file string: "none"
 2  xsize frozen         32          1       1024
 3   xoff frozen          0       -512        512
```

```
 4    fft frozen          1          0          1
 5 radial frozen          0          0          1

sherpa> psf0.file=test_model.dat
sherpa> show psf0
fpsf1d[psf0]
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   file string: "test_model.dat"
 2  xsize frozen         38          1         38
 3   xoff frozen          0        -19         19
 4    fft frozen          1          0          1
 5 radial frozen          0          0          1

sherpa> instrument = psf0
```

5.27 FPSF2D — FPSF — PSFFROMFILE

A 2-D file-based PSF instrument model.

`FPSF` is a file-based PSF model that represents the point-spread function, a redistribution function that maps photon spatial locations to image bins.

The input PSF FITS image is used to convolve (fold) a given source model. This convolution can be performed using either Fast Fourier Transforms (FFTs, the default), or the sliding cell technique (see the parameter `fft`). If the length of one axis is N and the length of the kernel axis is $M (= 2 \times \texttt{xsize} + 1)$, then the computation time for the sliding cell goes as $N \times M$, *i.e.*, for large kernels the best choice is using the FFT. A rough rule-of-thumb for 2-D fits is to use the FFT if $M_1 \times M_2 > 100$.

Note that the PSF is automatically renormalized upon entry. Renormalization is done by summing over all image pixels, regardless of the setting of xsize and ysize.

The following example will serve to clarify the meanings of the remaining parameters.

Assume that the PSF is provided in a FITS image (`file`) of size $N_x \times N_y$. N_x and N_y may be much larger than the PSF size in bins. An optimally sized sub-image of size $()xsize \times ysize$ can be extracted and used in the PSF convolution process.

If `xoff` = `yoff` = `0`, the kernel sub-image is extracted from the center of the original image contained in `file`. The user may find that changing `xoff` and/or `yoff` can result in his or her being able to extract a smaller optimally sized sub-image if, say, the PSF is too asymmetric to be fit easily into a centered rectangle. Note that actual PSF image may be outside the default extracted PSF image.

Note that the kernel centroid must always be at the center of the extracted sub-image! Otherwise, systematic shifts will occur in best-fit positions of point sources, *etc.*

See the documentation on the `INSTRUMENT` command (Section 1.46).

Number	Name	Description
1	`file`	input FITS file name
2	`xsize`	x-width of the subset region of kernel PSF file to use in convolution
3	`ysize`	y-width of the subset region of kernel PSF file to use in convolution
4	`xoff`	x-direction offset
5	`yoff`	y-direction offset
6	`fft`	convolution type: 1 = FFT / 0 = sliding cell

Table 5.31: FPSF Parameters

5.28 FRMF1D — FRMF

A 1-D file-based response matrix model.

`FRMF` is a file-based RMF model that represents the redistribution matrix which maps photon energy to, *e.g.*, PHA bin.

Note that in *Sherpa* analyses, the energy-space binning of the RMF (found typically in the `ENERG_LO` and `ENERG_HI` columns of the `MATRIX` extension of the RMF file) does not have to match the analogous energy-space binning of a matching ARF. (This is also true if instead of energy, the RMF is defined as a function of wavelength.)

See the documentation on the `INSTRUMENT` command (Section 1.46). See also the `RSP` (Section 5.51) and `FARF1D` (Section 5.24) instrument models.

Number	Name	Description
1	`rmf`	RMF file name

Table 5.32: FRMF Parameters

5.29 GAUSS1D — GAUSS

1-D unnormalized Gaussian function. Integration ON.

An unnormalized 1-D Gaussian model:

$$f(x) = A \exp\left[-f\left(\frac{x - x_o}{F}\right)^2\right] \tag{5.43}$$

The constant $f = 2.7725887 = 4\log 2$ relates the full-width at half-maximum F to the Gaussian σ so that F=sqrt(8log2)*sigma.

See also the 1-D normalized Gaussian function, `NGAUSS1D`, Section 5.42.

Number	Name	Description
1	`fwhm`	full-width at half-maximum F
2	`pos`	mean position x_{o}
3	`ampl`	amplitude A

Table 5.33: GAUSS1D Parameters

5.30 GAUSS2D

2-D unnormalized Gaussian function. Integration OFF.

A 2-D Gaussian model:

$$f(x,y) = f(r) = A \exp\left[-f\left(\frac{r}{F}\right)^2\right] \tag{5.44}$$

where

$$r(x,y) = \frac{\sqrt{x_{\mathrm{new}}^2(1-\epsilon)^2 + y_{\mathrm{new}}^2}}{1-\epsilon} \tag{5.45}$$

$$x_{\mathrm{new}} = (x - x_{\mathrm{o}})\cos\theta + (y - y_{\mathrm{o}})\sin\theta \tag{5.46}$$

and

$$y_{\mathrm{new}} = (y - y_{\mathrm{o}})\cos\theta - (x - x_{\mathrm{o}})\sin\theta \tag{5.47}$$

The constant $f = 2.7725887 = 4\log 2$ relates the full-width at half-maximum F to the Gaussian σ.

Number	Name	Description
1	`fwhm`	full-width at half-maximum F
2	`xpos`	x mean position x_{o}
3	`ypos`	y mean position y_{o}
4	`ellip`	ellipticity ϵ
5	`theta`	angle of ellipticity θ
6	`ampl`	amplitude A at (x_o, y_o)

Table 5.34: GAUSS2D Parameters

5.31 GRIDMODEL

N-D user-specified amplitude model. Integration OFF.

Number	Name	Description
1	`file`	input filename
2	`norm`	normalization

Table 5.35: GRIDMODEL Parameters

With `GRIDMODEL`, an N-dimensional data array may be input and used directly as a model. The data array must be of the same size as the source data and have the same binning.

Prior to CIAO 3.0, it was necessary to use `GRIDMODEL` when inputting an exposure map for use in spatial fitting; this is no longer true. Example 2 compares the CIAO 3.0 and 2.3 syntax; see the `FEXPMAP` command, Section 5.25, for further information.

Examples:

1. Input a data array as a model:

```
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GRIDMODEL[gm]
sherpa> gm.file = "data/example1.dat"
sherpa> THAW gm.norm
sherpa> SOURCE = gm
```

   In this example, the model component `GRIDMODEL` is established and assigned the name gm. Then, the input data array file is input using this model's `file` parameter. This model component, thus defined using an input data array, can then be used just as any other model component may be used. For example, here the model's normalization is thawed and then the model is used to define a source model expression for fitting.

2. Input an exposure map for use with a dataset (this no longer uses the `GRIDMODEL` command):

```
sherpa> DATA data/data.fits
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> FEXPMAP[emap]
sherpa> emap.file = "data/data_expmap.fits"
sherpa> INSTRUMENT = emap
```

   In this example, the instrument model component `FEXPMAP` is established, the exposure map file is input using this model's file parameter, and the instrument model stack is set for dataset number 1.

   The CIAO 2.3 equivalent of this command is:

```
sherpa> DATA data/data.fits
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> GRIDMODEL[gm]
sherpa> gm.file = "data/data_expmap.fits"
sherpa> EXPMAP 1 = gm
```

5.32 HIGHPASS — STEPHI1D

1-D step function. Integration OFF. The command `HIGHPASS` is equivalent.

A 1-D step model:

$$f(x) \;=\; A \qquad \text{if } x > x_{\rm cut}$$

and

$$f(x) \;=\; 0 \qquad \text{otherwise} \tag{5.49}$$

Number	Name	Description
1	xcut	cut-off x_cut
2	ampl	amplitude A

Table 5.36: STEPHI1D Parameters

5.33 HUBBLE — REYNOLDS

Hubble-Reynolds profile. Integration OFF. The `REYNOLDS` model is equivalent.

A 2-D Hubble-Reynolds model:

$$f(x,y) \;=\; f(r) \;=\; \frac{A}{\left[\sqrt{\left(\frac{x_{\rm maj}}{r_{\rm maj}}\right)^2 + \left(\frac{x_{\rm min}}{r_{\rm min}}\right)^2} + 1\right]^2} \tag{5.50}$$

where

$$r_{\rm maj} = r_0 \tag{5.51}$$

$$r_{\min} = (1-\epsilon)r_0 \quad (5.52)$$

$$x_{\mathrm{maj}} = (x-x_{\mathrm{o}})\cos\theta + (y-y_{\mathrm{o}})\sin\theta \quad (5.53)$$

and

$$x_{\min} = -(x-x_{\mathrm{o}})\sin\theta + (y-y_{\mathrm{o}})\cos\theta \quad (5.54)$$

Number	Name	Description
1	`r0`	core radius r_0
2	`xpos`	x mean position x_{o}
3	`ypos`	y mean position y_{o}
4	`ellip`	ellipticity ϵ
5	`theta`	angle of ellipticity θ
6	`ampl`	amplitude A

Table 5.37: HUBBLE Parameters

5.34 JDPILEUP

A CCD pileup model developed by John Davis of MIT.

The 1D pileup model can be applied to model the 1D Chandra spectrum obtained in the imaging mode. It should not be used for Chandra grating or 2D image data. This model is used for fitting energy spectra only.

Subsequent use of Powell and Simplex optimization methods is desirable when fitting data with pileup model.

Several model parameters describe pileup:

`n, f` and `g0`: The values of n, f and g0 should remain frozen; The full discussion of these parameters is presented in Davis (2001).

`alpha:`

The value of the parameter alpha should be allowed to vary. alpha parameterizes "grade migration" in the detector, and represents the probability, per photon count greater than one, that the piled event is not rejected by the spacecraft software as a "bad event". Specifically, if n photons are piled together in a single frame, the probability of them being retained (as a single photon event with their summed energy) is given by alpha(n-1). In reality, the alpha parameter should be a photon-energy-dependent and detector-chip-dependent matrix; for simplicity, the jdpileup model assumes a constant value.

`ftime:`

The ftime and parameter should be set to the value given in the header keyword EXPTIME of the event file. (Note that EXPTIME is used instead of TIMEDEL because the latter includes the transfer time, which ftime should not.)

`fracexp:`

The fracexp parameter should be set to the value given in the header keyword FRACEXPO of the ARF file:

Number	Name	Description
1	`alpha`	grade migration; probability of a good grade when two photons pile together
2	`g0`	probability of grade 0 assignment
3	`f`	fraction of flux falling into the pileup region
4	`n`	number of detection cells
5	`ftime`	frame time [seconds] (keyword `EXPTIME` in the event file)
6	`fracexp`	fractional exposure that the point source experienced while dithering on the chip (keyword `FRACPROB` in the ARF file)

Table 5.38: JDPILEUP Parameters

The pileup model does not work for pileup in dispersed grating spectra. The model was designed for imaging pileup, including pileup by the gratings in zeroth order.

For more information, see the Sherpa thread on Using a Pileup Model and the Event Pileup in Charge Coupled Devices page by John E. Davis.

Examples:

1. This example illustrate the setting of the pileup model.

```
sherpa> DATA spectrum.pi
sherpa> INSTRUMENT = RSP[myinst]("source.rmf", "source.arf")
sherpa> POW[p]
sherpa> SOURCE = p
sherpa> JDPILEUP[jdp]
sherpa> PILEUP = jdp
sherpa> show jdp
jdpileup[jdp]  (integrate: off)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1  alpha thawed        0.5          0          1
 2     g0 frozen          1     1e-120          1
 3      f thawed       0.95        0.9          1
 4      n frozen          1     1e-120        100
 5  ftime frozen      3.241     1e-120          5                   sec
 6fracexp frozen      0.987          0          1
sherpa> fit
```

In this example, data is input and a power-law model component is established and then defined as the source model for fitting. The command `JDPILEUP[jdp]` establishes the `JDPILEUP` model component and assigns it the name `jdp`. The command `PILEUP = jdp` then defines this pileup model for use in fitting.

2. After fitting, display the pileup fractions with `SHOW PILEUP`:

```
sherpa> SHOW PILEUP
   1: 0.224696  0.541445
   2: 0.263513  0.304025
   3: 0.206024  0.113808
   4: 0.120808  0.031952
   5: 0.0566711  0.00717652
   6: 0.0221538  0.00134322
   7: 0.00742312  0.000215494
   8: 0.00217637  3.02503e-05
   9: 0.000567189  3.77462e-06
   10: 0.000133035  4.23895e-07
   *** pileup fraction: 0.458555
```

Each row represents the number of photons per CCD frame. The maximum number of rows is 20; here, there were no more than 10 photons piled together per frame. The left column represents the percentage of frames with the stated number of photons: here, 22.47% of frames contained a single photon, 26.35% contained two photons piled together, *etc.* The right column indicates the percentage of observed counts with the stated number of photons: here, 54.14% of recorded counts were comprised of one photon, *etc.*. The total pileup fraction is the sum of the right column, excluding the first row: here, 45.85% of observed counts actually contain two or more photons piled together.

5.35 KERNEL

Defines an expression to be used as a kernel in 1D source model convolution.

```
sherpa> KERNEL [<dataset range> | ALLSETS [ID]] = <modelExpr>
```

<dataset range> = # (or more generally #:#,#:#, *etc.*) such that # specifies a dataset number and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

The model expression, <`modelExpr`>, is an algebraic combination of one or more of the following elements:

```
{<sherpa_modelname> | <sherpa_modelname>[modelname] |
 <modelname> | <model_stack> | <nested_model>}
```

along with numerical values. The following operators are recognized: + - * / ( ) { }. See the `CREATE` (Section 1.23) command for further information.

Note that:

- The documentation on *Sherpa Models* (Section 5) contains a summary list, and descriptions, of the models that are available within *Sherpa*, which include models from *XSPEC*, v. 11.3.
- By default, if the model expression includes a model component that has not previously been established, *Sherpa* will prompt for the initial parameter values for that model component. This prompting can be turned off using the PARAMPROMPT OFF (Section 1.58) command.

To reset the kernel model stack, issue the command:

```
sherpa> KERNEL [<dataset range> | ALLSETS] =
```

The KERNEL model stack is used to convolve source model expressions. As such, it has many uses, of which two are the following:

- It can be used instead of an INSTRUMENT model stack if the instrumental response is representable by analytic functions (*e.g.*, by a Gaussian).
- It can be used to model circumstellar smearing of photon spectra; these spectra may then be further smeared by the observing instrument.

Note that the parameter values of a KERNEL model stack are frozen in fits; the kernel shape cannot be optimized.

The KERNEL model stack is evaluated on the same data grid as the SOURCE model stack. In general, the data grid will not be the correct grid for convolution; in the case of Gaussian smoothing, the convolved spectrum may exhibit, *e.g.*, spectral lines that are both dilated *and translated.* To prevent translation, the user must set an origin for the convolution kernel using the command:

```
sherpa> KERNEL.ORIGIN [<dataset range> | ALLSETS] = <coordinate>
```

The example below will help illustrate this point.

Examples:

1. Define SOURCE and KERNEL model stacks; define the origin of the convolution kernel; and fit:

```
sherpa> SOURCE = GAUSS[modela]
modela.fwhm parameter value [19.3451]
modela.pos parameter value [804]
modela.ampl parameter value [126]
sherpa> KERNEL = NGAUSS[modelb]
modelb.fwhm parameter value [19.3451] 20
modelb.pos parameter value [804]
modelb.ampl parameter value [2594.62] 1:1:1
sherpa> KERNEL.ORIGIN = 804
sherpa> FIT
```

```
 LVMQT: V2.0
 LVMQT: initial statistic value = 578.346
 LVMQT: final statistic value = 64.8424 at iteration 6
        modela.fwhm  25.3426
        modela.pos  807.882
        modela.ampl  150.302
sherpa> KERNEL =
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 294.76
 LVMQT: final statistic value = 64.3803 at iteration 7
        modela.fwhm  32.3536
        modela.pos  807.863
        modela.ampl  117.826
```

The convolution kernel is a normalized Gaussian, with integrated area set to 1 and full-width half-max of 20. (For a normalized Gaussian, the integrated area equals the amplitude.) The kernel origin is set to be the centroid of the normalized Gaussian (`modelb.pos`). Note the differences in the values of `modela.fwhm` and `modela.ampl` between each fit.

5.36 LINEBROAD

Line broadening profile. Integration ON.

A line-broadening profile:

$$f(\lambda) = \frac{2Ac}{\pi\lambda v\sin i}\left[1-\left(\frac{\Delta\lambda}{\lambda_{\rm rest}}\frac{c}{v\sin i}\right)^2\right] \qquad (5.55)$$

where

$$\Delta\lambda = \lambda - \lambda_{\rm rest}. \qquad (5.56)$$

Number	**Name**	**Description**
1	`rest`	rest wavelength $\lambda_{\rm rest}$ (A)
2	`vsini`	rotation velocity $v\sin i$ (km/s)
3	`ampl`	amplitude A

Table 5.39: LINEBROAD Parameters

5.37 LORENTZ1D — LORENTZ

1-D normalized Lorentzian function. Integration ON.

A normalized 1-D Lorentz model:

$$f(x) = \frac{A}{\pi} \frac{\frac{F}{2}}{\left(\frac{F}{2}\right)^2 + (x - x_o)^2} \tag{5.57}$$

with

$$\int_{-\infty}^{+\infty} dx f(x) = A \tag{5.58}$$

Number	Name	Description
1	`fwhm`	full-width at half-maximum F
2	`pos`	mean position x_o
3	`ampl`	amplitude A

Table 5.40: LORENTZ1D Parameters

5.38 LORENTZ2D

2-D unnormalized Lorentzian function. Integration OFF.

An unnormalized 2-D Lorentz model:

$$f(x,y) = f(r) = A \frac{\left(\frac{F}{2}\right)^2}{\left(\frac{F}{2}\right)^2 + r^2} \tag{5.59}$$

where

$$r(x,y) = \frac{\sqrt{x_{\mathrm{new}}^2 (1-\epsilon)^2 + y_{\mathrm{new}}^2}}{1-\epsilon} \tag{5.60}$$

$$x_{\mathrm{new}} = (x - x_o) \cos\theta + (y - y_o) \sin\theta \tag{5.61}$$

and

$$y_{\mathrm{new}} = (y - y_o) \cos\theta - (x - x_o) \sin\theta \tag{5.62}$$

Number	Name	Description
1	fwhm	full-width at half-maximum F
2	xpos	x mean position $x_{\rm o}$
3	ypos	y mean position $y_{\rm o}$
4	ellip	ellipticity ϵ
5	theta	angle of ellipticity θ
6	ampl	amplitude A at (x_o, y_o)

Table 5.41: LORENTZ2D Parameters

5.39 LORPOW2D — BETA2D

Another name for the `BETA2D` model. See Section 5.10.

5.40 LOWPASS — STEPLO1D

1-D step function. Integration OFF. The command `LOWPASS` is equivalent.

A 1-D step model:

$$f(x) \;=\; A \qquad \text{if } x < x_{\rm cut} \tag{5.63}$$

$$f(x) \;=\; 0 \qquad \text{otherwise} \tag{5.64}$$

Number	Name	Description
1	xcut	cut-off x_cut
2	ampl	amplitude A

Table 5.42: STEPLO1D Parameters

5.41 NBETA

1-D normalized beta function. Integration ON.

A normalized 1-D beta function appropriate for use fitting line profiles:

$$f(x) \;=\; \frac{A}{\left[1+\frac{(x-x_0)^2}{w^2}\right]^{-\alpha}} \tag{5.65}$$

Number	Name	Description
1	`pos`	line centroid x_0
2	`width`	line width w
3	`index`	index α
4	`ampl`	line amplitude A

Table 5.43: NGAUSS1D Parameters

5.42 NGAUSS1D

1-D normalized Gaussian function. Integration ON.

A normalized 1-D Gaussian model:

$$f(x) = \frac{A}{\sqrt{\frac{\pi}{f}}F} \exp\left[-f\left(\frac{x-x_o}{F}\right)^2\right] \tag{5.66}$$

with

$$\int_{-\infty}^{+\infty} dx f(x) = A \tag{5.67}$$

The constant $f = 2.7725887 = 4\log 2$ relates the full-width at half-maximum F to the Gaussian σ so that F=sqrt(8log2)*sigma.

See also the 1-D unnormalized Gaussian function, `GAUSS1D`, Section 5.29.

Number	Name	Description
1	`fwhm`	full-width at half-maximum F
2	`pos`	mean position x_o
3	`ampl`	amplitude A

Table 5.44: NGAUSS1D Parameters

5.43 PILEUP

Defines the pileup model expression to be used for fitting a 1D dataset.

```
sherpa> PILEUP [<dataset range> | ALLSETS] = <pileupModelExpr>
```

<dataset range> = # (or more generally #:#,#:#, *etc.*) such that # specifies a dataset number and #:# represents an inclusive range of datasets; one may specify multiple inclusive ranges by separating them with commas. The default dataset is dataset 1.

In CIAO 3.0 *Sherpa*, there is one pileup model available: `JDPILEUP`, developed by John Davis at MIT. The pileup model does not work for pileup in dispersed spectra. Rather, the model was designed for imaging pileup, including pileup by the gratings in zeroth order. This model is used for fitting energy spectra only.

Note that:

- To apply the `JDPILEUP` model, an instrument model with ARF must be defined, and the ARF must have a uniform grid.
- The parameters of the `JDPILEUP` model (in particular, α) themselves do not indicate how much pileup occurred during the observation. To retrieve this information, use the command `SHOW PILEUP`, based on the `PRINT_KERNEL` function of the fitting application *ISIS*. See the example below for an explanation of the output of `SHOW PILEUP`.
- By default, if the model instance of `JDPILEUP` has not previously been established, *Sherpa* will prompt for its initial parameter values. This prompting can be turned off using the PARAMPROMPT OFF (Section 1.58) command.

To reset the pileup model stack, issue the command:

```
sherpa> PILEUP [<dataset range> | ALLSETS] =
```

Examples:

1. Define a pileup model for use:

```
sherpa> DATA data/source.pi
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: statistical errors specified in the PHA file.
         These are currently IGNORED.  To use them, type:
         READ ERRORS "<filename>[cols CHANNEL,STAT_ERR]" fitsbin
sherpa> PARAMPROMPT OFF
Model parameter prompting is off
sherpa> INSTRUMENT = RSP[myinst]("data/source.rmf", "data/source.arf")
The inferred file type is ARF.  If this is not what you want, please
specify the type explicitly in the data command.
sherpa> POW[p]
sherpa> SOURCE = p
sherpa> JDPILEUP[jdp]
sherpa> PILEUP = jdp
sherpa> FIT
```

In this example, data is input and a power-law model component is established and then defined as the source model for fitting. The command JDPILEUP[jdp] establishes the JDPILEUP model component and assigns it the name jdp. The command PILEUP = jdp then defines this pileup model for use in fitting.

2. After fitting, display the pileup fractions with SHOW PILEUP:

```
sherpa> SHOW PILEUP
   1: 0.224696  0.541445
   2: 0.263513  0.304025
   3: 0.206024  0.113808
   4: 0.120808  0.031952
   5: 0.0566711  0.00717652
   6: 0.0221538  0.00134322
   7: 0.00742312  0.000215494
   8: 0.00217637  3.02503e-05
   9: 0.000567189  3.77462e-06
   10: 0.000133035  4.23895e-07
   *** pileup fraction: 0.458555
```

Each row represents the number of photons per CCD frame. The maximum number of rows is 20; here, there were no more than 10 photons piled together per frame. The left column represents the percentage of frames with the stated number of photons: here, 22.47% of frames contained a single photon, 26.35% contained two photons piled together, *etc.* The right column indicates the percentage of observed counts with the stated number of photons: here, 54.14% of recorded counts were comprised of one photon, *etc.*. The total pileup fraction is the sum of the right column, excluding the first row: here, 45.85% of observed counts actually contain two or more photons piled together.

5.44 POISSON

Poisson function. Integration OFF.

A model expressing the ratio of two Poisson distributions of mean mu, one for which the random variable is x, and the other for which the random variable is equal to mu itself:

$$f(x) = A\frac{\frac{\mu^x}{x!}\exp(-\mu)}{\frac{\mu^\mu}{\mu!}\exp(-\mu)} = A\frac{\mu!}{x!}\mu^{x-\mu} \tag{5.68}$$

Number	Name	Description
1	mean	mean μ
2	ampl	amplitude A

Table 5.45: POISSON Parameters

5.45 POLYNOM1D — POLY

1-D polynomial function. Integration ON.

A 1-D polynomial of order ≤ 8:

$$f(x) = \sum_{i=0}^{8} c_i (x - x_{\rm off})^i \tag{5.69}$$

where the coefficients c_i are the parameters numbered $i+1$, and $x_{\rm off}$ is parameter number 10.

Note that there is a degeneracy in the parameters, so it is recommended to set at least one of c_0 or $x_{\rm off}$ to zero and freeze it; *thawing both may lead to unpredicted results.*

Note also that all coefficients except c_0 are default frozen, so that the default polynomial model is a constant.

Number	Name	Description
1	`c0`	coefficient c_0
2	`c1`	coefficient c_1
3	`c2`	coefficient c_2
4	`c3`	coefficient c_3
5	`c4`	coefficient c_4
6	`c5`	coefficient c_5
7	`c6`	coefficient c_6
8	`c7`	coefficient c_7
9	`c8`	coefficient c_8
10	`offset`	offset for x $x_{\rm off}$

Table 5.46: POLYNOM1D Parameters

5.46 POLYNOM2D

2-D polynomial function. Integration ON.

A 2-D polynomial model of order ≤ 2 :

$$f(x,y) = c_0 + c_{\rm x1}x + c_{\rm x2}x^2 + c_{\rm y1}y + c_{\rm y2}y^2 + c_{\rm x1,y1}xy + c_{\rm x1,y2}xy^2 + c_{\rm x2,y1}x^2y + c_{\rm x2,y2}x^2y^2 \tag{5.70}$$

Number	Name	Description
1	`constant`	constant coefficient
2	`cx1`	coefficient $c_{\rm x1}$
3	`cx2`	coefficient $c_{\rm x2}$
4	`cy1`	coefficient $c_{\rm y1}$
5	`cy2`	coefficient $c_{\rm y2}$

6	`cx1y1`	coefficient $c_{\mathrm{x1,y1}}$
7	`cx1y2`	coefficient $c_{\mathrm{x1,y2}}$
8	`cx2y1`	coefficient $c_{\mathrm{x2,y1}}$
9	`cx2y2`	coefficient $c_{\mathrm{x2,y2}}$

Table 5.47: POLYNOM2D Parameters

5.47 POWLAW1D

1-D power law. Integration ON.

A 1-D power-law model:

$$f(x) \;=\; A\left(\frac{x}{x_{\mathrm{ref}}}\right)^{-\gamma} \tag{5.71}$$

The variable x is assumed to be > 0.

Number	**Name**	**Description**
1	`gamma`	power law photon index γ
2	`ref`	normalization reference point x_{ref}
3	`ampl`	amplitude A

Table 5.48: POWLAW1D Parameters

5.48 PTSRC1D

A 1-D file-based point-source fitting model.

`PTSRC1D` is a file-based model that may be used in fitting sources, to determine, *e.g.*, if they are point-like or extended.

This model cannot be used as an instrument model.

ASCII and FITS formats are allowed for the input model `file`. The current implementation requires that the bin size of the data and the `PTSRC1D` model file are the same. The number of bins can be different.

The model has several parameters, which can be used to adjust the model array to be used in fitting.

`"xsize"` parameter defines the number of bins extracted from the file to be used in the model.

`xoff = 0` parameters defines the center of the array and it should be left at 0 in CIAO 3.2 version of the model.

"`norm`" parameter is thawed by definition and it is defined as a total number of counts (values) in the model. The best-fit value for a point-like source will be approximately equal to the number of detected counts from the source.

"`xpos`" parameter indicates where the center of the sub-array is located in the extracted model array, for example it should be set to the location of the source centroid if the input file is a gaussian function. Note that the initial value of the `xpos` is estimated using the input dataset.

Number	Name	Description
1	`file`	input file name (FITS or ASCII)
2	`xsize`	x-full-width of the subset region of the model file
3	`xoff`	x-direction offset
4	`xpos`	x-position of the centroid in the data coordinates
5	`norm`	normalization

Table 5.49: PTSRC1D Parameters

Examples:

1. The example below shows the initial parameter values for the `PTSRC1D` model. Note that the default `xsize` value is set to 32 before the data file is read in. After reading the data file the `xsize` is set to the size of the array in the model file.

```
sherpa> ptsrc1d[p1]
sherpa> show p1
ptsrc1d[p1]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   file string: "none"
 2  xsize frozen         32          1       1024
 3   xoff frozen          0       -512        512
 4   xpos thawed       12.5       12.5      197.5
 5   norm thawed          1          0       1000

shrpa> p1.file="model.dat"
sherpa> show p1
ptsrc1d[p1]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   file string: "model.dat"
 2  xsize frozen         38          1         38
 3   xoff frozen          0        -19         19
 4   xpos thawed       12.5       12.5      197.5
 5   norm thawed          1          0       1000

sherpa> source 1 = p1
sherpa> fit
```

5.49 PTSRC2D —PTSRC

A 2-D file-based point-source fitting model.

`PTSRC` is a file-based model that may be used in fitting image data, to determine, *e.g.*, if they are point-like or extended.

This model cannot be used as an instrument model.

FITS formats are allowed for the input model `file.` The current implementation requires that the image bins of the data and the `PTSRC2D` model file are the same.

Assume that the PSF is provided in a FITS image (`file`) of size $N_x \times N_y$. N_x and N_y may be much larger than the PSF size in pixels. An optimally sized sub-image of size $xsize \times ysize$ can speed up the computation of model amplitudes.

If `xoff` = `yoff` = `0`, the sub-image is extracted from the center of the original image contained in file. The user may find that changing `xoff` and/or `yoff` can result in his or her being able to extract a smaller optimally sized sub-image if, say, the PSF is too asymmetric to be fit easily into a centered rectangle.

If `norm` is thawed its best-fit value for a point-like source will be approximately equal to the number of detected counts from the source.

The parameters `xpos` and `ypos` indicate where the center of the sub-image is to be placed in the data image that is to be analyzed, *i.e.*, they should be set to the location of the source centroid. Note that the initial values of the `xpos` and `ypos` are estimated using the input dataset.

Number	**Name**	**Description**
1	`file`	input FITS file name
2	`xsize`	x-width of the subset region of kernel file to use in convolution
3	`ysize`	y-width of the subset region of kernel file to use in convolution
4	`xoff`	x-direction offset
5	`yoff`	y-direction offset
6	`xpos`	x-position of the centroid in data image coordinates
7	`ypos`	y-position of the centroid in data image coordinates
8	`norm`	normalization defined as a sum over the model image

Table 5.50: PTSRC Parameters

Examples:

1. The example below shows the initial parameter values for the `PTSRC2D` model. Note that the default `xsize` and `ysize` values are set to 32. After reading the data file these parameter can be adjusted to expand the image and include most of the PSF fraction. Notice after image command indicates the fraction of PSF included in the sub-image.

```
sherpa> ptsrc2d[p2]
sherpa> show p2
ptsrc2d[p2]  (integrate: on)
```

```
   Param   Type      Value        Min        Max                 Units
   -----   ----      -----        ---        ---                 -----
1   file string: "none"
2  xsize frozen         32          1       1024
3  ysize frozen         32          1       1024
4   xoff frozen          0       -512        512
5   yoff frozen          0       -512        512
6   xpos thawed      128.5        0.5      256.5
7   ypos thawed      128.5        0.5      256.5
8   norm thawed          1          0       1000

sherpa> p2.file=psf2.fits
sherpa> show p2
ptsrc2d[p2]  (integrate: on)
   Param   Type      Value        Min        Max                 Units
   -----   ----      -----        ---        ---                 -----
1   file string: "psf2.fits"
2  xsize frozen         32          1       1024
3  ysize frozen         32          1       1024
4   xoff frozen          0       -512        512
5   yoff frozen          0       -512        512
6   xpos thawed      128.5        0.5      256.5
7   ypos thawed      128.5        0.5      256.5
8   norm thawed          1          0       1000
sherpa> image p2
NOTE: PSF fraction for (xsize,ysize): FRAC = 0.984733

sherpa> p2.xsize=64
sherpa> p2.ysize=64
sherpa> image p2
```

5.50 REYNOLDS — HUBBLE

Another name for the HUBBLE model. See Section 5.33.

5.51 RSP1D —RSP

A 1-D instrument model.

RSP is used to model instrument response. It has two parameters (as listed below): ARF and RMF filenames. The ARF file contains an ancillary response vector representing effective area as a function of photon energy, while the RMF file contains the redistribution matrix which maps photon energy to, *e.g.*, PHA bin.

There are several things to be aware of:

- It is not necessary to specify both an ARF and RMF file to do analysis; for instance, the user can analyze data using only an ARF.
- In *Sherpa* analyses, the energy-space binning of the ARF (found typically in the `ENERG_LO` and `ENERG_HI` columns of the `SPECRESP` extension of the ARF file) does not have to match the analogous energy-space binning of the RMF. (This is also true if instead of energy, the ARF is defined as a function of wavelength.)
- The explicit use of `RSP` is only necessary if the filenames are not given in the header of the input PHA data file.

See the documentation on the `INSTRUMENT` command (Section 1.46). See also the `FARF1D` (Section 5.24) and `FRMF1D` (Section 5.28) instrument models.

Number	Name	Description
1	`rmf`	RMF file name
2	`arf`	ARF file name

Table 5.51: RSP Parameters

5.52 RSP2D

A 2-D instrument model utilizing an exposure map and point-spread function.

`RSP2D` is used to model instrument response with an exposure map and point-spread function (PSF).

Note the following:

- The exposure map image must have the same size as the data image, and it must have the same plate-scale (*i.e.*, bins in each image must have the same size in arc-seconds, *etc.*).
- The exposure map and PSF images need *not* have the same size, but both must have the same plate-scale.

See the documentation on the `INSTRUMENT` command (Section 1.46). See also the `FARF2D` (Section 5.25) and `FPSF2D` (Section 5.27) instrument models, for a fuller explanation of the parameters.

Number	Name	Description
1	`psffile`	PSF file name
2	`xsize`	x-full-width of the subset region of the PSF file to use in convolution
3	`ysize`	y-full-width of the subset region of the PSF file to use in convolution
4	`xoff`	x-direction offset
5	`yoff`	y-direction offset
6	`fft`	convolution type: 1 = FFT / 0 = sliding cell
7	`empfile`	exposure map file name

Table 5.52: RSP2D Parameters

5.53 SCHECHTER

Schechter function. Integration OFF.

An integrated 1-D Schechter model:

$$f(x_i) \;=\; A\left(\frac{x_i}{x_{\rm ref}}\right)^{\alpha}\exp\left(-\frac{x_i}{x_{\rm ref}}\right)\frac{x_{i+1}-x_i}{x_{\rm ref}} \tag{5.72}$$

Number	Name	Description
1	`alpha`	power coefficient α
2	`ref`	reference value $x$ref
3	`norm`	normalization factor A

Table 5.53: SCHECHTER Parameters

5.54 SHEXP10

Exponential function, base 10. Integration OFF.

An exponential model, base 10:

$$f(x) \;=\; A10^{C(x-x_{\rm off})} \tag{5.73}$$

Number	Name	Description
1	`offset`	offset $x_{\rm off}$
2	`coeff`	coefficient C
3	`ampl`	amplitude A

Table 5.54: SHEXP10 Parameters

5.55 SHEXP

Exponential function. Integration OFF.

An exponential model:

$$f(x) \;=\; A\exp[C(x-x_{\rm off})] \tag{5.74}$$

Number	Name	Description
1	`offset`	offset x_off

2	`coeff`	coefficient C
3	`ampl`	amplitude A

Table 5.55: SHEXP Parameters

5.56 SHLOG10

Logarithm function, base 10. Integration OFF.

A base-10 logarithm model:

$$f(x) = A \log_{10}[C(x - x_{\rm off})] \tag{5.75}$$

Number	Name	Description
1	`offset`	offset $x_{\rm off}$
2	`coeff`	coefficient C
3	`ampl`	amplitude A

Table 5.56: SHLOG10 Parameters

5.57 SHLOGE — SHLOG

Natural logarithm function. Integration OFF.

A natural logarithm model:

$$f(x) = A \log[C(x - x_{\rm off})] \tag{5.76}$$

Number	Name	Description
1	`offset`	offset $x_{\rm off}$
2	`coeff`	coefficient C
3	`ampl`	amplitude A

Table 5.57: SHLOGE Parameters

5.58 SIN

Sine function. Integration OFF.

A 1-D sine model:

$$f(x) = A\sin\left(\frac{2\pi(x - x_{\rm off})}{P}\right) \tag{5.77}$$

Number	Name	Description
1	`period`	period P, in units of x
2	`offset`	x offset $x_{\rm off}$
3	`ampl`	amplitude A

Table 5.58: SIN Parameters

5.59 SQRT

Square root function. Integration OFF.

A 1-D square-root model:

$$f(x) = A\sqrt{x - x_{\rm off}} \tag{5.78}$$

Number	Name	Description
1	`offset`	offset $x_{\rm off}$
2	`ampl`	amplitude A

Table 5.59: SQRT Parameters

5.60 STEPHI1D — HIGHPASS

Another name for the `HIGHPASS` model. See Section 5.32.

5.61 STEPLO1D — LOWPASS

Another name for the `LOWPASS` model. See Section 5.40.

5.62 TAN

Tangent function. Integration OFF.

A 1-D tangent model:

$$f(x) = A \tan\left(\frac{2\pi(x - x_{\rm off})}{P}\right) \quad (5.79)$$

Number	Name	Description
1	`period`	period P, in units of x
2	`offset`	x offset $x_{\rm off}$
3	`ampl`	amplitude A

Table 5.60: TAN Parameters

5.63 TPSF1D

A 1-D TCD-model-based instrument model.

`TPSF1D` is a model-based instrument model. It utilizes the predefined Gaussian, box, and top hat functions of the TCD library.

Note that Top Hat function cannot be used in version 3.2

The convolution can be performed using either Fast Fourier Transforms (FFTs, the default), or the sliding cell technique (see the parameter `fft`). If the axis length is N and the length of the kernel axis is $M (= 2 \times \texttt{xsize} + 1)$, then the computation time for the sliding cell goes as $N \times M$, *i.e.* for large kernels the best choice is using the FFT. A rough rule-of-thumb for 1-D fits is to use the FFT if $M > 10$.

In *Sherpa* version 3.0.2, a new parameter is introduced: `radial`. If set to 1 the kernel array will be extended and its values reflected across the edge boundary. The resultant function will be symmetric. The default value (set to 0) should be used in version 3.2.

See the documentation on the `INSTRUMENT` command (Section 1.46).

Number	**Name**	**Description**
1	`xsize`	x-axis kernel width (pixels)
2	`nsigma`	sigma kernel width (Gaussian only)
3	`funcTyp`	Gaussian (1) / box (2) / top-hat (3)
4	`fft`	convolution type: 1 = FFT / 0 = sliding cell
5	`radial`	radial profile: 1 = yes / 0 = no

Table 5.61: TPSF1D Parameters

Examples:

1. This example illustrate the definition of the `TPSF1D` model.

```
sherpa> tpsf1d[t1]
sherpa> show t1
tpsf1d[t1]
```

```
    Param   Type      Value        Min        Max                Units
    -----   ----      -----        ---        ---                -----
 1  xsize frozen          4          1       1024
 2 nsigma frozen          2      1e-02        100
 3funcTyp frozen          1          0          7
 4    fft frozen          1          0          1
 5 radial frozen          0          0          1
 The Function Type is:  Gaussian.
```

The parameter funcTyp in t1 model indicate that the Gaussian function will be used. The 1 sigma width of the Gaussian is set to xsize=4 bins and the total size of the Gaussian kernel is set to 2 sigma.

5.64 TPSF2D — TPSF — PSFFROMTCD — PSF

A 2-D TCD-model-based instrument model.

TPSF is a model-based 2D instrument model. It utilizes the predefined Gaussian, box, and top hat functions of the TCD library. It can represent the point-spread function (PSF), a redistribution function that maps photon spatial locations to image bins.

Note that the top hat function cannot be used in version 3.2

The convolution can be performed using either Fast Fourier Transforms (FFTs, the default), or the sliding cell technique (see the parameter fft). If the length of one axis is N and the length of the kernel axis is $M(= 2 \times \texttt{xsize} + 1)$, then the computation time for the sliding cell goes as $N \times M$, *i.e.* for large kernels the best choice is using the FFT. A rough rule-of-thumb for 2-D fits is to use the FFT if $M_1 \times M_2 > 100$.

See the documentation on the INSTRUMENT command (Section 1.46).

Number	Name	Description
1	xsize	x-axis 1 sigma width (pixels)
2	ysize	y-axis 1 sigma width (pixels)
3	nsigma	kernel width in sigma (Gaussian only)
4	funcTyp	Gaussian (1) / box (2) / top-hat (3)
5	fft	convolution type: 1 = FFT / 0 = sliding cell

Table 5.62: TPSF Parameters

Examples:

1. This example illustrates the definition of the TPSF1D model.

```
sherpa> tpsf2d[t2]
sherpa> show t2
tpsf2d[t2]
```

```
   Param   Type      Value        Min        Max                 Units
   -----   ----      -----        ---        ---                 -----
1  xsize frozen          4          1       1024
2  ysize frozen          4          1       1024
3 nsigma frozen          2      1e-02        100
4funcTyp frozen          1          0          7
5    fft frozen          1          0          1
The Function Type is:  Gaussian.
```

The parameter **funcTyp** in **t2** model indicate that the Gaussian function will be used. The 1 sigma width of the Gaussian is set to xsize=ysize=4 pixels and the total size of the Gaussian kernel is set to 2 sigma.

5.65 USERMODEL

User implemented model.

It is possible for the user to create and implement his or her own model, own optimization method, and own statistic function within *Sherpa*. The *User Models, Statistics, and Methods Within Sherpa* chapter (Chapter 7) of the *Sherpa Reference Manual* has more information on this topic.

The tar file **sherpa_user.tar.gz** contains the files needed to define the **usermethod**, e.g Makefiles and Implementation files, plus example files, and it is available from the Sherpa threads page: Data for Sherpa Threads

5.66 XS<xspecname>

XSpec model functions.

Sherpa includes the "additive" and "multiplicative" models of *XSpec* version 11.3. To use these models, simply prefix the original *XSpec* name by "xs"; for example, the name xsphabs represents the *XSpec* phabs model. To define a composite *XSpec* model such as

$$phabs * apec \tag{5.80}$$

do the following:

```
sherpa> xsphabs[abs1]
sherpa> xsapec[gas1]
sherpa> source 1 = abs1 * gas1
```

(Of course, one can use names other than "abs1" and "gas1" to represent these model components.)

Note that XSPEC models are always integrated over a bin and so require binned data.

5.66.1 Important Note:

XSpec models based on physical processes (*e.g.* line models such as raymond or absorption models such as wabs) assume that the dataspace is defined in keV. On the other hand *Sherpa* models are always calculated based on the input data scale. Thus when *XSpec* models are combined with *Sherpa* models, the user should be careful to ensure that both components have the same dataspace units; otherwise, calculated model amplitudes may be incorrect.

These models also expect that the x-values will always be energy bins. When the analysis setting is using non-energy bins (*e.g.*, `ANALYSIS WAVE`) and an XSPEC model is defined, *Sherpa* converts the bins to energy before sending them to the XSPEC model. After the XSPEC model finishes, *Sherpa* converts back to the original units. *Sherpa* also scales the model values appropriately (*e.g.*, if counts/keV came out of the XSPEC model, and *Sherpa* is working with wavelength bins, then *Sherpa* scales the output of the XSPEC model to counts/Angstrom).

5.66.2 Unavailable XSPEC Models and Commands

The following XSPEC components are NOT included in CIAO 3.4:

- the bkn2pow (three-segment broken power law) and kerrd (optically thick extreme-Kerr disk) models
- the xset command: this means that users cannot currently modify certain XSPEC internal switches.
- the atable, etable, and mtable models

5.66.3 Available XSPEC Models

The available *XSpec* models are listed below. Please see either the ahelp page for each model (*e.g.* "ahelp xsabsori") or the XSpec User's Guide for more information about each of these models. Note that the ahelp files describe the version of the XSpec model included in CIAO, while the XSpec User's Guide may reference a newer version with different options.

<xspecname>	Description
absori	Ionized absorber
acisabs	Decay in the ACIS quantum efficiency
apec	APEC thermal plasma model
bapec	APEC thermal plasma model with velocity broadening as a free parameter
bbody	Blackbody spectrum
bbodyrad	Blackbody spectrum with norm proportional to surface area
bexrav	E-folded broken power law reflected from neutral matter
bexriv	E-folded broken power law reflected from ionized matter
bknpower	Broken power law
bmc	Comptonization by relativistically moving matter
bremss	Thermal bremsstrahlung
bvapec	APEC thermal plasma model with variable abundances and velocity broadening as a free parameter
c6mekl	6th-order Chebyshev polynomial DEM using mekal

c6pmekl	Exponential of 6th-order Chebyshev polynomial DEM using mekal
c6pvmkl	Variable abundance version of c6pmekl
c6vmekl	Variable abundance version of c6mekl
cabs	Compton scattering (non-relativistic)
cemekl	Multi-temperature mekal
cevmkl	Multi-temperature vmeka
cflow	Cooling flow model
compbb	Comptonized blackbody spectrum after Nishimura et al. (1986)
compls	Comptonization spectrum after Lamb and Sanford (1979)
compst	Comptonization spectrum after Sunyaev and Titarchuk (1980)
comptt	Comptonization spectrum after Titarchuk (1994)
constant	Energy-independent multiplicative factor
cutoffpl	Power law with high energy exponential cutoff
cyclabs	Cyclotron absorption line
disk	Disk model
diskbb	Multiple blackbody disk model
diskline	Line emission from relativistic accretion disk
diskm	Disk model with gas pressure viscosity
disko	Modified blackbody disk model
diskpn	Accretion disk around a black hole
dust	Dust scattering out of the beam
edge	Absorption edge
equil	Equilibrium ionization collisional plasma model from Borkowski
expabs	Low-energy exponential cutoff
expdec	Exponential decay
expfac	Exponential factor
gabs	Multiplicative gaussian absorption line
gaussian	Simple gaussian line profile
gnei	Generalized single ionization NEI plasma model
grad	GR accretion disk around a black hole
grbm	Gamma-ray burst model
highecut	High energy cutoff
hrefl	Simple reflection model good up to 15 keV
laor	Line from accretion disk around a black hole
lorentz	Lorentzian line profile
meka	Mewe-Gronenschild-Kaastra thermal plasma (1992)
mekal	Mewe-Kaastra-Liedahl thermal plasma (1995)
mkcflow	Cooling flow model based on mekal
nei	Simple nonequilibrium ionization plasma model
notch	Notch line absorption
npshock	Plane-parallel shock with ion and electron temperatures
nsa	Spectra in the X-ray range (0.05-10 keV) emitted from a hydrogen atmosphere of a neutron star.
nteea	Pair plasma model
pcfabs	Partial covering fraction absorption
pegpwrlw	Power law with pegged normalization
pexrav	Exponentially cutoff power law reflected from neutral matter
pexriv	Exponentially cutoff power law reflected from ionized matter
phabs	Photo-electric absorption

plabs	Absorption model with power law dependence on energy
plcabs	Cutoff power law observed through dense, cold matter
posm	Positronium continuum
powerlaw	Simple photon power law
pshock	Constant temperature, plane-parallel shock plasma model
pwab	Extension of partial covering fraction absorption into a power-law distribution of covering fraction
raymond	Raymond-Smith thermal plasma
redden	IR/optical/UV extinction from Cardelli *et al.* (1989)
redge	Recombination edge
refsch	E-folded power law reflected from an ionized relativistic disk
sedov	Sedov model with electron and ion temperatures
smedge	Smoothed absorption edge
spline	Spline multiplicative factor
srcut	Synchrotron radiation from cutoff electron distribution
sresc	Synchrotron radiation from escape-limited electron distribution
SSS_ice	Einstein Observatory SSS ice absorption
step	Step function convolved with gaussian
TBabs	Calculates the absorption of X-rays by the ISM
TBgrain	Calculates the absorption of X-rays by the ISM with variable hydrogen to H2 ratio and grain parameters
TBvarabs	Calculates the absorption of X-rays by the ISM, allowing user to vary all abundances, depletion factors, and grain properties
uvred	UV reddening
vapec	APEC thermal plasma model with variable abundances
varabs	Photoelectric absorption with variable abundances
vbremss	Thermal bremsstrahlung spectrum with variable H/He
vequil	Ionization equilibrium collisional plasma model with variable abundances
vgnei	Non-equilibrium ionization collisional plasma model with variable abundances
vmcflow	Cooling flow model based on vmekal
vmeka	M-G-K thermal plasma with variable abundances
vmekal	M-K-L thermal plasma with variable abundances
vnei	Non-equilibrium ionization collisional plasma model with variable abundances
vnpshock	Plane-parallel shock plasma model with separate ion and electron temperatures and variable abundances
vphabs	Photoelectric absorption with variable abundances
vpshock	Constant temperature, plane-parallel shock plasma model with variable abundances
vraymond	Raymond-Smith thermal plasma with variable abundances
vsedov	Sedov model with separate ion and electron temperatures and variable abundances
wabs	Photoelectric absorption (Morrison and McCammon)
wndabs	Photoelectric absorption with low energy window
xion	Reflected spectra of a photo-ionized accretion disk or ring
zbbody	Redshifted blackbody
zbremss	Redshifted thermal bremsstrahlung
zedge	Redshifted absorption edge
zgauss	Redshifted gaussian
zhighect	Redshifted high energy cutoff
zpcfabs	Redshifted partial covering absorption
zphabs	Redshifted photoelectric absorption
zpowerlw	Redshifted power law
zTBabs	Calculates the absorption of X-rays by the ISM for modeling redshifted absorption. Does not include a dust component.
zvarabs	Redshifted photoelectric absorption with variable abundances

zvfeabs	Redshifted absorption with variable iron abundance
zvphabs	Redshifted photoelectric absorption with variable abundances
zwabs	Redshifted "Wisconsin absorption"
zwndabs	Redshifted photoelectric absorption with low energy window

Chapter 6

Sherpa Fit Statistics Functions

6.1 Summary of Available Statistics

<statisticname>	**Description**
BAYES	A Bayesian maximum likelihood function.
CASH	A maximum likelihood function.
{CHI CVAR | CHI PARENT}	χ^2 statistic with constant variance computed from the counts data.
CHI DVAR	χ^2 statistic with variance computed from the data.
CHI GEHRELS	χ^2 statistic with the Gehrels variance function.
CHI MVAR	χ^2 statistic with variance computed from model amplitudes.
{CHI PARENT | CHI CVAR}	χ^2 statistic with constant variance computed from the counts data.
CHI PRIMINI	χ^2 statistic with Primini variance function.
CSTAT	A maximum likelihood function: XSPEC implementation of CASH.
USERSTAT	User implemented statistic.

Table 6.1: Summary of Statistics

In this chapter we describe each of the *Sherpa* fit statistics, in alphabetical order. Similar information about each statistic is available within *Sherpa* with the command `AHELP -W <statisticname>`, where **<statisticname>** is the name of a statistic.

These descriptions are meant to serve as a brief reference.

6.2 BAYES

A Bayesian maximum likelihood function.

In conventional fits involving background data, it is the best-fit background model amplitude in bin i, B_i, that is used when one attempts to ascertain the source model amplitude S_i. While this is the most probable value of B_i, it is not the only possible value, and S_i may change if all possible values of B_i, weighted by their likelihood, are taken into account. This can be done using Bayes' Theorem. Below, we assume familiarity with the basics of Bayesian statistical methodology; a reader who is not familiar with these basics should consult, *e.g.*, the review text by Loredo (1992, Statistical Challenges in Modern Astronomy, ed. Feigelson & Babu, 275).

For simplicity, assume that there are two fitting 'regions' (in time, space, or both), denoted 'off-source' and 'on-source,' with $N_{i,B}$ and $N_{i,S}$ counts respectively. Counts are sampled from the Poisson distribution, and the best way to assess the quality of model fits is the use the Poisson likelihood function. Within the on-source region, the Poisson likelihood of the sum $S_i + B_i$ is

$$p(N_{i,S}|S_i, B_i) \ = \ \frac{(S_i + B_i)^{N_{i,S}}}{N_{i,S}!} \exp(-S_i - B_i)\,. \tag{6.1}$$

We can relate this likelihood to the Bayesian posterior density for S_i and B_i using Bayes' Theorem:

$$p(S_i, B_i|N_{i,S}) \ = \ p(S_i|B_i)p(B_i)p(N_{i,S}|S_i, B_i)/p(D)\,. \tag{6.2}$$

The factor $p(S_i|B_i)$ is the Bayesian prior probability for the source model amplitude, which is assumed to be constant, and $p(D)$ is an ignorable normalization constant. The prior probability $p(B_i)$ is treated differently; we can specify it using the posterior probability for B_i off-source:

$$p(B_i) \ = \ (A\frac{(AB_i)^{N_{i,B}}}{N_{i,B}!}) \exp(-AB_i)\,, \tag{6.3}$$

where A is an "area" factor that rescales the number of predicted background counts B_i to the off-source region.

IMPORTANT: this formula is derived assuming that the background is constant as a function of spatial area, time, etc. If the background is not constant, the Bayes function should not be used.

To take into account all possible values of B_i, we integrate, or marginalize, the posterior density $p(S_i, B_i|N_{i,S})$ over all allowed values of B_i:

$$p(S_i|N_{i,S}) \ = \ \int_0^\infty p(S_i, B_i|N_{i,S})dB_i\,. \tag{6.4}$$

For the constant background case, this integral may be done analytically. We do not show the final result here; see Loredo. The function $-\log\, p(S_i|N_{i,S})$ is minimized to find the best-fit value of S_i. The magnitude of this function depends upon the number of bins included in the fit and the values of the data themselves. Hence one cannot analytically assign a 'goodness-of-fit' measure to a given value of this function. Such a measure can, in principle, be computed by performing Monte Carlo simulations. One would repeatedly sample new datasets from the best-fit model, and fit them, and note where the observed function minimum lies within the derived distribution of minima. (The ability to perform Monte Carlo simulations is a feature that will be included in a future version of *Sherpa*.)

6.2.1 Note on Background Subtraction

The background should not be subtracted from the data when this function is used The background only needs to be specified, as in this example:

```
sherpa> DATA 1 source.data
sherpa> BACK 1 background.data
sherpa> SOURCE 1 = [sourcemodel]
sherpa> STATISTIC BAYES
sherpa> FIT
...
```

To compare the best-fit source model with the data in a plotting environment, one would have to subtract the background from the raw counts data, as in this example:

```
...
sherpa> FIT
...
sherpa> SUBTRACT
sherpa> PLOT FIT
```

Otherwise, the plotted best-fit model and data will differ by approximately the background amplitude.

If the number of counts in each bin is high (> 5), the likelihood is approximately proportional to $\exp(-\chi^2/2)$ Hence `UNDERFLOW` errors can occur if the initial parameter estimate is too far from the location of a local likelihood maximum, or if parameter step sizes are set too large. (This is because during intermediate steps of the computation of $-\log\, p(S_i|N_{i,S})$, absolute likelihoods must be computed, unlike for the case of the `CASH` statistic, where the log-likelihood terms never need to be exponentiated.) To avoid these errors, it may be necessary to use a χ^2 statistic to find the approximate solution before using the Bayesian maximum likelihood function.

Examples:

1. Specify the fitting statistic and then confirm it has been set. The method is then changed from "Levenberg-Marquardt" (the default), since this statistic does not work with that algorithm.

```
sherpa> STATISTIC BAYES
sherpa> SHOW STATISTIC
Statistic:           Bayes
sherpa> METHOD POWELL
```

6.3 CASH

A maximum likelihood function.

Counts are sampled from the Poisson distribution, and so the best way to assess the quality of model fits is to use the product of individual Poisson probabilities computed in each bin i, or the likelihood $\mathcal{L}$:

$$\mathcal{L} = \prod_i \frac{M_i^{D_i}}{D_i!} \exp(-M_i) \tag{6.5}$$

where $M_i = S_i + B_i$ is the sum of source and background model amplitudes, and D_i is the number of observed counts, in bin i.

The `CASH` statistic (Cash 1979, ApJ 228, 939) is derived by (1) taking the logarithm of the likelihood function, (2) changing its sign, (3) dropping the factorial term (which remains constant during fits to the same dataset), and (4) multiplying by two:

$$C = 2 \sum_i [M_i - D_i \log M_i] \tag{6.6}$$

The factor of two exists so that the change in `CASH` statistic from one model fit to the next, ΔC, is distributed approximately as $\Delta\chi^2$ when the number of counts in each bin is high (> 5). One can then in principle use ΔC instead of $\Delta\chi^2$ in certain model comparison tests. However, unlike χ^2, the `CASH` statistic may be used regardless of the number of counts in each bin.

The magnitude of the `CASH` statistic depends upon the number of bins included in the fit and the values of the data themselves. Hence one cannot analytically assign a 'goodness-of-fit' measure to a given value of the `CASH` statistic. Such a measure can, in principle, be computed by performing Monte Carlo simulations. One would repeatedly sample new datasets from the best-fit model, and fit them, and note where the observed `CASH` statistic lies within the derived distribution of `CASH` statistics. (The ability to perform Monte Carlo simulations is a feature that will be included in a future version of *Sherpa*.)

6.3.1 Note on Background Subtraction

The background should not be subtracted from the data when this statistic is used. It should be modeled simultaneously with the source, as in this example:

```
sherpa> DATA source.data
sherpa> BACK background.data
sherpa> SOURCE = [source model]
sherpa> BG = [background model]
sherpa> STATISTIC CASH
sherpa> FIT
```

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CASH
sherpa> SHOW STATISTIC
Statistic:          Cash
```

6.4 χ^2 statistics

Chi-square statistic.

The chi-square statistic is

$$\chi^2 = \sum_i \frac{[N_{i,S} - B_i(x_i, p_B) - S_i(x_i, p_S)]^2}{\sigma_i^2} \tag{6.7}$$

where $N_{i,S}$ is the total number of observed counts in bin i of the on-source region; $B_i(x_i, p_B)$ is the number of predicted background model counts in bin i of the on-source region (zero for background-subtracted data), rescaled from bin i of the off-source region, and computed as a function of the model argument x_i (*e.g.*, energy or time) and set of background model parameter values p_B; $S_i(x_i, p_S)$ is the number of predicted source model counts in bin i, as a function of the model argument x_i and set of source model parameter values p_S; and σ_i is the error in bin i.

The options for assigning σ_i are described in the documentation for `CHI DVAR (CHIDVAR)`, `CHI GEHRELS (CHIGEHRELS)`, `CHI MVAR (CHIMVAR)`, and `CHI PRIMINI (CHIPRIMINI)`. In each of these files, $N_{i,B}$ is the total number of observed counts in bin i of the off-source region; A_B is the off-source 'area,' which could be the size of the region from which the background is extracted, or the length of a background time segment, or a product of the two, *etc.*; and A_S is the on-source 'area.'

In the analysis of PHA data, A_B is the product of the `BACKSCAL` and `EXPTIME` FITS header keyword values, provided in the file containing the background data. A_S is computed similarly, from keyword values in the source data file.

Note that in the current version of *Sherpa*, it is assumed that there is a one-to-one mapping between a given background region bin and a given source region bin. For instance, in the analysis of PHA data, it is assumed that the input background counts spectrum is binned in exactly the same way as the input source counts spectrum, and any filter applied to the source spectrum automatically applied to the background spectrum. This means that currently, the user cannot, *e.g.*, specify arbitrary background and source regions in two dimensions and get correct results. This will be changed in a future version of *Sherpa*.

(However, this limitation only applies when analyzing background data that have been entered with the `BACK` command. One can always enter the background as a separate dataset and jointly fit the source and background regions.)

6.4.1 CHI CVAR — CHI PARENT

Chi-square statistic with constant variance computed from the counts data.

In some applications, analysts have seen fit to assume that the variance is constant for each bin. For this choice of statistic, the variance is assumed to be the mean number of counts, or

$$\sigma_i^2 = \frac{1}{N} \sum_{j=1}^{N} N_{j,S} + \left(\frac{A_S}{A_B}\right)^2 N_{j,B} \,, \tag{6.8}$$

where N is the number of on-source (and off-source) bins included in the fit. The background term appears only if a background region is specified *and* background subtraction is done.

See `CHISQUARE` for more information, including definitions of the quantities shown above.

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CHI CVAR
sherpa> SHOW STATISTIC
Statistic:          Chi-Squared Constant Variance
```

6.4.2 CHI DVAR

Chi-square statistic with variance computed from the data.

If the number of counts in each bin is large (> 5), then the shape of the Poisson distribution from which the counts are sampled tends asymptotically towards that of a Gaussian distribution, with variance

$$\sigma_i^2 = N_{i,S} + \left(\frac{A_S}{A_B}\right)^2 N_{i,B} \,. \tag{6.9}$$

The background term appears only if a background region is specified *and* background subtraction is done.

See `CHISQUARE` for more information, including definitions of the quantities shown above.

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CHI DVAR
sherpa> SHOW STATISTIC
Statistic:          Chi-Squared Data Variance
```

6.4.3 CHI GEHRELS

Chi-square statistic with the Gehrels variance function.

This is the *Sherpa* default statistic.

If the number of counts in each bin is small (< 5), then we cannot assume that the Poisson distribution from which the counts are sampled has a nearly Gaussian shape. The standard deviation (*i.e.*, the square-root of the variance) for this low-count case has been derived by Gehrels (1986):

$$\sigma_{i,S} = 1 + \sqrt{N_{i,S} + 0.75}. \tag{6.10}$$

Higher-order terms have been dropped from the expression; it is accurate to approximately one percent. If one does not perform background subtraction, then $\sigma_i = \sigma_{i,S}$; otherwise, one may use standard error propagation to estimate that

$$\sigma_i^2 = \sigma_{i,S}^2 + \left(\frac{A_S}{A_B}\right)^2 \sigma_{i,B}^2 \,. \tag{6.11}$$

The background term appears only if a background region is specified *and* background subtraction is done. See `CHISQUARE` for more information, including definitions of the quantities shown above.

6.4.4 Note on Background Subtraction

We have not determined the accuracy of the latter expression, thus the user should proceed with caution when subtracting background from the raw data when using this statistic. An approach preferable to background subtraction is to model the background and data simultaneously.

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CHI GEHRELS
sherpa> SHOW STATISTIC
Statistic:            Chi-Squared Gehrels
```

6.4.5 CHI MVAR

Chi-square statistic with variance computed from model amplitudes.

This statistic is equivalent to `CHI DVAR`, Section 6.4.2, except that the variance is estimated using the background and source model amplitudes rather than the observed counts data:

$$\sigma_i^2 = S_i + \left(\frac{A_S}{A_B}\right)^2 B_{i,\mathrm{off}} \,, \tag{6.12}$$

where $B_{i,\mathrm{off}}$ is the background model amplitude in bin i of the off-source region. See `CHISQUARE` for more information, including definitions of the quantities shown above.

6.4.6 Note on Background Subtraction

The background should not be subtracted from the data when this statistic is used. `CHI MVAR` underestimates the variance when fitting background-subtracted data.

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CHI MVAR
sherpa> SHOW STATISTIC
Statistic:          Chi-Squared Model Variance
```

6.4.7 CHI PARENT — CHI CVAR

The statistic `CHI CVAR` is equivalent. See Section 6.4.1 for futher information.

6.4.8 CHI PRIMINI

Chi-square statistic with Primini variance function.

The χ^2 statistic is a biased estimator of model parameters (unlike the likelihood functions). In an attempt to remove this bias, Kearns, Primini, & Alexander (1995, ADASS IV, 331) use a scheme dubbed 'Iterative Weighting' (IW; see Wheaton *et al.* 1995, ApJ 438, 322), in which

$$\sigma_i^2 \; = \; S(i, x_i, pS^{j-1}) + \left(\frac{A_N}{A_B}\right)^2 B_{\rm off}(i, x_i, pB^{j-1}) \, , \tag{6.13}$$

where j is the number of iterations that have been carried out in the fitting process, $B_{\rm off}$ is the background model amplitude in bin i of the off-source region, and pS^{j-1} and pB^{j-1} are the set of source and background model parameter values derived during the iteration previous to the current one.

In addition to reducing parameter estimate bias, it can be used even when the number of counts in each bin is small (< 5), although the user should proceed with caution.

6.4.9 Note on Background Subtraction

The background should not be subtracted from the data when this statistic is used. `CHI PRIMINI` underestimates the variance when fitting background-subtracted data.

See `CHISQUARE` for more information, including definitions of the quantities shown above.

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CHI PRIMINI
WARNING: with CHI PRIMINI, displayed error bars will only be
         correct after a fit is performed with the current filter.
sherpa> SHOW STATISTIC
Statistic:           Chi-Squared Primini
```

6.5 CSTAT

A maximum likelihood function.

The CSTAT statistic is equivalent to the *XSPEC* implementation of the Cash statistic.

Counts are sampled from the Poisson distribution, and so the best way to assess the quality of model fits is to use the product of individual Poisson probabilities computed in each bin i, or the likelihood $\mathcal{L}$:

$$\mathcal{L} \;=\; \prod_i \frac{M_i^{D_i}}{D_i!} \exp(-M_i) \tag{6.14}$$

where $M_i = S_i + B_i$ is the sum of source and background model amplitudes, and D_i is the number of observed counts, in bin i.

The CSTAT statistic (Cash 1979, ApJ 228, 939) is derived by (1) taking the logarithm of the likelihood function, (2) changing its sign, (3) dropping the factorial term (which remains constant during fits to the same dataset), (4) adding an extra data-dependent term, and (4) multiplying by two:

$$C \;=\; 2\sum_i \left[M_i - D_i + D_i(\log D_i - \log M_i)\right] \tag{6.15}$$

The factor of two exists so that the change in CSTAT statistic from one model fit to the next, ΔC, is distributed approximately as $\Delta\chi^2$ when the number of counts in each bin is high (> 5). One can then in principle use ΔC instead of $\Delta\chi^2$ in certain model comparison tests. However, unlike χ^2, the CSTAT statistic may be used regardless of the number of counts in each bin.

The advantage of CSTAT over *Sherpa*'s implementation of CASH is that one can assign an approximate 'goodness-of-fit' measure to a given value of the CSTAT statistic, *i.e.*, the observed statistic, divided by the number of degrees of freedom, should be of order 1 for good fits.

6.5.1 Note on Background Subtraction

The background should not be subtracted from the data when this statistic is used. It should be modeled simultaneously with the source, as in this example:

```
sherpa> DATA source.data
sherpa> BACK background.data
sherpa> SOURCE = [source model]
sherpa> BG = [background model]
sherpa> STATISTIC CSTAT
sherpa> FIT
```

Examples:

1. Specify the fitting statistic and then confirm it has been set.

```
sherpa> STATISTIC CSTAT
sherpa> SHOW STATISTIC
Statistic:           Cstat
```

6.6 USERSTAT

User implemented statistic.

It is possible for the user to create and implement his or her own model, own optimization method, and own statistic function within *Sherpa*. The *User Models, Statistics, and Methods Within Sherpa* chapter (Chapter 7) of the *Sherpa Reference Manual* has more information on this topic.

The tar file **sherpa_user.tar.gz** contains the files needed to define the **userstat**, e.g Makefiles and Implementation files, plus example files, and it is available from the Sherpa threads page: Data for Sherpa Threads

Chapter 7

User Models, Statistics, and Methods Within *Sherpa*

It is possible for the user to create and include their own model, optimization method, and statistic function within *Sherpa*. Examples, which are discussed below, are provided in the `sherpa_user.tar.gz` file available from the *Sherpa* threads page. The file should be unpacked into its own directory. It contains a `README` file with pointers to the relevant files and directories.

7.1 User Models

The most common addition to *Sherpa* are user-defined models. These allow you to fit your data using your own model. There are three ways to write user models:

1. using `S-Lang`.

2. using a compiled language (`C`, `C++`, `FORTRAN`) with a `S-Lang` interface to *Sherpa*.

3. using a compiled language to interface to *Sherpa*.

7.1.1 Using `S-Lang`

User models can be written purely in `S-Lang`, as described in `ahelp slang usermodel`. This approach is recommended when the user model is not very complicated, and you do not already have a version written in a compiled language such as `C`, `C++`, or `FORTRAN`. It is also a good way to test out a model, since you can move to using a compiled version later on if necessary.

Advantages

- Models are quick to develop since you only need to edit a text file (no compile or linking stage is necessary).
- The model can be re-loaded without needing to re-start *Sherpa*, which is useful for testing and adding new features.
- Models can be given a useful name, for example you can register a model called `mymodel` and then use it as you would any other *Sherpa* model. For example

```
sherpa> source = mymodel[bob]
```

- Multiple `S-Lang` user models can be used in the same session. So for instance you can say

```
sherpa> source = mymodel1[bob] + mymodel2[fred]
```

  where `mymodel1` and `mymodel2` are `S-Lang` user models.

Disadvantages

- You have to be able to write the model in `S-Lang`.
- For complicated models you may be concerned about the speed of the `S-Lang` code.

Examples of writing 1D and 2D models in `S-Lang` can be found in the `model_1d_noint/pure_slang_version/` and `model_2d_noint/pure_slang_version/` directories.

7.1.2 Using a compiled language and S-Lang

For complicated models, where execution speed is important, or for existing models which may already be written in a compiled language, it is possible to create a `S-Lang` module that lets *Sherpa* call this model.

This is the recommended way to use compiled models from *Sherpa*. The alternative is to use the `ascfitUserModel` interface which is described in section 7.1.3.

Advantages

- Models can be given a useful name, for example you can register a model called `mymodel` and then use it as you would any other *Sherpa* model. For example

```
sherpa> source = mymodel[bob]
```

- Multiple user models can be used in the same session. So for instance you can say

```
sherpa> source = mymodel1[bob] + mymodel2[fred]
```

where `mymodel1` and `mymodel2` are user models.

- Users do not need to tell the model how many parameters are by the model (as they do with the `ascfitUserModel` interface).
- The model can be called directly from the *Sherpa* prompt, which allows easy testing of the results.

Disadvantages

- The interface to the compiled code requires some knowledge of the `S-Lang` API - in particular the Intrinsic Functions section of the S-Lang Library C Programmer's Guide.

  Note that the Programmer's Guide refers to **version 2** of `S-Lang` whereas `CIAO 3.4` uses **version 1.4.9**. See the upgrade guide for more information on the changes between the two versions of `S-Lang`.

Examples of writing 1D and 2D models in a compiled language with a `S-Lang` interface can be found in the `c_slang_version/`, `cc_slang_version/`, and `fortran_slang_version/` directories of `model_1d_noint/`, and `model_2d_noint/`.

7.1.3 Using a compiled language (the `ascfitUserModel` interface)

The `ascfitUserModel` interface is the original method provided by *Sherpa* for writing user models. It only allows one user model to be loaded into a *Sherpa* session, the model will be called `usermodel`, and it is used slightly differently to the other *Sherpa* models since users need to tell *Sherpa* how many model parameters there are.

The `S-Lang` interface (section 7.1.2) is the preferred way to add compiled code as a user model, but this interface is still provided.

Advantages

- The interface requires no knowledge of `S-Lang`, only the language used to write the model.

Disadvantages

- Only one model can be loaded into a *Sherpa* session.
- The number of model parameters must be supplied by the user of the model
- It is harder to test the model from Sherpa than when the `S-Lang` interface is used.

Examples of writing 1D and 2D models using the `ascfitUserModel` interface can be found in the `c_version/` and `fortran_version/` directories of `model_1d_noint/`, and `model_2d_noint/`.

7.2 User Method

An optimization method contains the definition of the algorithm which is responsible for finding a minimum for the fit statistic. This is done by varying the fit parameters within their ranges. Different optimization methods use different algorithms to determine how the parameter space is searched for the minima.

The purpose of having a user-defined optimization method is to integrate your own local algorithm definition within the *Sherpa* fitting and modeling environment, for your own use. Unlike user models (section 7.1), the only way to write such a method is to use the `ascfitUserMethod` interface. An example method is provided in the `method/` directory.

7.3 User Statistic

A fit statistic is the measure you use to compare your predicted model data to your real measured data. It is by minimizing the fit statistic that you find a good fit for your model. An example of a fit statistic is `CHI GEHRELS`.

The purpose of having a user defined fit statistic is to integrate your own local algorithm definition within the *Sherpa* fitting and modeling environment, for your own use. Unlike user models (section 7.1), the only way to write such a method is to use the `ascfitUserStatistic` interface. An example method is provided in the `stat/` directory.

Chapter 8

S-lang in *Sherpa*

The S-Lang programming language can be used to manipulate or create data for plotting; see "**ahelp slang**" for more information on S-Lang. This chapter describes the S-lang functions that duplicate and extend the *Sherpa* commands.

All of the information provided here is available within CIAO and *Sherpa* from the **ahelp** system; "**ahelp -c sherpa**" returns a list of all the help files relating to *Sherpa* .

S-lang commands are **case sensitive** and must be written in lowercase.

8.1 compute_errors

Module function to estimate errors for an array of data

```
Array_Type compute_errors(Array_Type)
```

Error Return Value: NULL

Argument:

(1) Array of values of type `Float_Type` or `Double_Type`

This function estimates the errors for each element of the input array given *Sherpa*'s current `STATISTIC` setting. (*Note that these data need not have been input into Sherpa!*) Thus there is an implicit assumption that the input array elements are counts. If the input array elements are not counts, then one could, *e.g.*, create a new S-Lang function to compute the errors properly.

This function is only appropriate for use with data-based variances: **chi gehrels**, **chi dvar**, and **chi parent**.

Note that currently, the input array cannot be of type `Integer_Type`. This will be fixed in a future version of the module. To check the type of the array, use the S-Lang function `_typeof`:

```
sherpa> a = [5,3,9]
sherpa> _typeof(a)
Integer_Type
sherpa> compute_errors(a)
NULL
```

One can get around this error by explicitly typecasting the array:

```
sherpa> a = typecast(a,Double_Type)
sherpa> foo = compute_errors(a)
sherpa> print(foo)
3.39792
2.93649
4.1225
```

See the related function `compute_statistic`.

8.2 compute_statistic

Module function to estimate a statistic given arrays of data, model amplitudes, and errors, *etc.*

```
Double_Type compute_statistic(Struct_Type)
Double_Type compute_statistic(Array_Type,Array_Type \
[,Array_Type[,Array_Type[,Array_Type]]]))
```

Error Return Value: NULL

Arguments:

(1) Structure containing data, model amplitude, *etc.*, arrays; or

(1) Array of observed data

(2) Array of predicted model amplitudes

(3) Array of estimated errors

(4) Array of systematic errors

(5) Array of statistical weights

This function computes the summed statistic value given *Sherpa*'s current `STATISTIC` setting. (*Note that the data et al. need not have been input into Sherpa!*)

If an error array is input, then note that the actual setting of χ^2 (*e.g.*, `CHI GEHRELS` or `CHI DVAR`) does not matter (since the input errors are used instead of error estimates) if the χ^2 statistic is used. Also note that the error array is ignored for likelihood-based statistics (*e.g.*, `CASH`).

On the other hand, if an errors array is not provided, and a data-based variance is used (`chi gehrels`, `chi dvar`, and `chi parent`), then errors are estimated via a call to the function `compute_errors`; there is thus an implicit assumption that the input data array elements are counts. If `chi mvar` is used, then the errors are set equal to the input predicted model amplitudes.

Examples:

1. Compute the statistic given a data and model array (with errors estimated):

```
sherpa> get_stat_expr
chi gehrels
sherpa> d = [55,53,59]
sherpa> m = [55.5,55.6,55.7]
sherpa> compute_statistic(d,m)
0.243771
```

Because an error array was not provided, errors were estimated using the `CHI GEHRELS` variance formula; then the total statistic was computed. Below, we explicit create and use an array of errors:

```
sherpa> e = [5.5,5.3,5.9]
sherpa> compute_statistic(d,m,e)
0.561761
sherpa> STATISTIC CHI DVAR
sherpa> compute_statistic(d,m,e)
0.561761
```

The version of χ^2 does not matter since the error estimates σ are provided.

8.3 create_model

Module functions to instantiate new source or instrument models.

```
Integer_Type create_model(String_Type[,String_Type])
Integer_Type create_inst_model(String_Type[,String_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Name of *Sherpa* model component (*e.g.*, `GAUSS`)

(2) ID applied to that component (*e.g.*, `g`)

See the related *Sherpa* command `CREATE` for more information.

NOTE: In CIAO 3.1 the definition of INSTRUMENT BACK is required for fitting PHA data if either background file or background models have been defined. INSTRUMENT BACK is set automatically when the PHA data file is input to *Sherpa*, however it is deleted if the NEW background file is input for a given data set. Thus the new INSTRUMENT BACK has to be defined on the command line before starting a simultaneous fit with the new background file.

Examples:

1. Instantiate a polynomial model, and name it `p`:

   ```
   sherpa> () = create_model("POLY","p")
   ```

2. Instantiate an `rsp` instrument model, and name it `a`:

   ```
   sherpa> () = create_inst_model("RSP","a")
   ```

3. Instantiate a Gaussian model with default ID:

   ```
   sherpa> () = create_model("GAUSS1D")
   sherpa> SHOW
   ...
   gauss1d[GAUSS1D]  (integrate: on)
       Param   Type      Value        Min        Max                 Units
       -----   ----      -----        ---        ---                 -----
    1   fwhm thawed         10 1.1755e-38 3.4028e+38
    2    pos thawed          0 -3.403e+38 3.4028e+38
    3   ampl thawed          1 -3.403e+38 3.4028e+38
   ```

8.4 dcounts

Calculates the sum of observed counts data for source or background datasets.

```
sherpa> [B]DCOUNTS [# [ID]] [{(<value>) | (<min>:<max>) | (<region descriptor>)} ]
```

`DCOUNTS` is used for summing source data counts, while `BDCOUNTS` is used for summing background data counts.

`#` specifies the dataset over which the source model is evaluated. The `ID` modifier is used only for summing background counts, and then if and only if the *Sherpa* state object variable `multiback` is set to 1, *i.e.*, if more than one background dataset is to be associated with a single source dataset. The `ID` modifier may be any unreserved string (*e.g.*, A, foo, *etc.*), *i.e.*, a string that is not a parsable command.

The default is to do summations for all datasets. The summation may be computed at one energy/wavelength, over a range of energies/wavelengths, or within a 2-D region, with the default being to compute the flux the total available range.

For 1-D data, if

- one energy (or wavelength or bin number) <value> is given, then the data counts is returned for the bin associated with that value.
- an energy or wavelength range is given with the <`min`> and <`max`> arguments, then the total summed data counts for the given range is returned.
- no value or range is specified, then the total summed data counts for the full range of the dataset is returned.

For 2-D data, if a region descriptor is given, then the summation of data counts within that region is returned; otherwise, the summation is carried out over the entire input image.

Tip: To perform background subtraction in *Sherpa*, the command `SUBTRACT`, Section 1.80, must be issued; this is in contrast to *XSPEC*, which performs background subtraction automatically.

The summation of data counts may be done using the *Sherpa*/S-Lang module functions `get_dcounts_sum` and `get_bdcounts_sum`.

Examples:

1. Calculate the summation of data counts over the full energy range:

```
sherpa> DCOUNTS
Observed counts for source dataset 1: 1688 counts
```

2. Calculate the summation of background data counts over the range 2 to 10 keV. If the areas of the source and background extraction regions are different (as manifested by differences in, *e.g.*, the `BACKSCAL` keywords in the headers of the source and background data files), then the number of expected background counts in the source region will also be shown.

```
sherpa> BDCOUNTS (2.0:10.0)
Observed counts for background dataset 1: 775 counts
     ...scaled to source region: 238.779 counts
```

3. Calculate the summation of data counts at a single energy (2.0 keV):

```
sherpa> DCOUNTS 1 (2.0)
Observed counts for source dataset 1: 23 counts
```

4. Calculate the summation of data counts over the energy range 2.0 to 4.0 keV:

```
sherpa> DCOUNTS 1 (2:4)
Observed counts for source dataset 1: 328 counts
```

5. Calculate the total flux within circles of radius 5 and 10 in a 2-D image (note, the quotes are necessary):

```
sherpa> DCOUNTS 1 "CIRCLE(247,257,5)"
Observed counts for source dataset 1: 1331 counts
sherpa> DCOUNTS 1 "CIRCLE(247,257,10)"
Observed counts for source dataset 1: 3068 counts
```

8.5 get

Summary of *Sherpa*/S-Lang module functions that retrieve settings or data.

The `get` functions of the *Sherpa*/S-Lang module:

- Retrieve numerical data *from Sherpa* so that, *e.g.*, they may be manipulated using S-Lang operators and/or functions. (These data may be put back into *Sherpa* using analogous `set` functions.)
- Retrieve *Sherpa* settings. (Quantities may be set using analogous `set` functions.)

Note that not all `get` functions have `set` function analogues: for instance, while it makes sense to retrieve, change, and set estimated errors, it makes little sense to put new arrays of residuals or predicted model amplitudes *into Sherpa*.

Name	**Description**
`get_data` | `get_back`	Retrieves amplitudes of source and background datasets
`get_fluxed_spectrum` | `get_back`	Retrieve a fluxed spectrum (counts divided by ARF)
`get_errors` | `get_berrors`	Retrieves error estimates of source and background datasets
`get_syserrors` | `get_bsyserrors`	Retrieves systematic errors of source and background datasets
`get_weights` | `get_bweights`	Retrieves statistical weights for source and background datasets
`get_mcounts` | `get_bmcounts`	Retrieves predicted source and background model counts amplitudes
`get_residuals` | `get_bresiduals`	Retrieves the fit residuals for source and background datasets
`get_delchi` | `get_bdelchi`	Retrieves the fit sigma residuals for source and background datasets
`get_ratio` | `get_bratio`	Retrieves the ratio of data to model for source and background datasets
`get_statistics` | `get_bstatistics`	Retrieves the contribution to the current statistic value from each bin, for source and background datasets

`get_source` | `get_bg`	Retrieves predicted source and background model photon amplitudes
`get_groups` | `get_bgroups`	Retrieves grouping arrays associated with source and background dataset
`get_quality` | `get_bquality`	Retrieves quality arrays associated with source and background dataset
`get_filter` | `get_bfilter`	Retrieves filter arrays associated with source and background datasets
`get_axes` | `get_baxes`	Retrieves the energy/wavelength/channel grid of source and background datasets
`get_energy_axes` | `get_energy_baxes`	Retrieves the energy grid of source and background datasets
`get_wave_axes` | `get_wave_baxes`	Retrieves the wavelength grid of source and background datasets
`get_raw_axes` | `get_raw_baxes`	Retrieves the raw channel grid of source and background datasets
`get_photon_axes` | `get_bphoton_axes`	Retrieves photon-space grids over which models are evaluated
`get_photon_energy_axes` | `get_photon_energy_baxes`	Retrieves photon-space energy grids over which models are evaluated
`get_photon_wave_axes` | `get_photon_wave_baxes`	Retrieves photon-space wavelength grids over which models are evaluated
`get_arf_axes` | `get_arf_baxes`	Retrieves the energy/wavelength grid of an ARF associated with source and background datasets
`get_analysis`	Retrieves the current analysis setting.
`get_coord`	Retrieves the current coord setting.
`get_fit` | `get_goodness`	Retrieves information about the quality of a fit
`get_statistic` | `get_bstatistic`	Retrieves the current value of the statistic comparing source and background data and model values
`get_record`	Returns a record of model parameter values at the end of each iteration of the fitting process
`get_flux` | `get_bflux`	Returns the unconvolved photon flux for source or background datasets
`get_pflux2d`	Returns photon fluxes in 2-D images
`get_eflux` | `get_beflux`	Returns the unconvolved energy flux for source or background datasets
`get_eflux2d`	Returns energy fluxes in 2-D images
`get_mcounts_sum` | `get_bmcounts_sum`	Returns the sum of convolved model counts in source and background datasets
`get_mcounts_sum2d`	Returns sums of model counts in 2-D images
`get_dcounts_sum` | `get_bdcounts_sum` | `get_net_counts_sum`	Returns the sum of observed counts in source and background datasets
`get_dcounts_sum2d`	Returns sums of observed counts in 2-D images
`get_eqwidth` | `get_beqwidth`	Returns the equivalent width of a line in source or background data
`get_par`	Retrieve model parameter values, *etc.*
`get_unc`	Retrieves parameter bounds
`get_proj`	Retrieves parameter bounds
`get_cov`	Retrieves parameter bounds

`get_intunc`	Retrieves parameter value and best-fit statistic arrays
`get_intproj`	Retrieves parameter value and best-fit statistic arrays
`get_regunc`	Retrieves parameter value and best-fit statistic arrays
`get_regproj`	Retrieves parameter value and best-fit statistic arrays
`get_dimension`	Retrieve the dimensionality of source data
`get_exptime` | `get_bexptime`	Retrieve source and background exposure times
`get_backscale` | `get_bbackscale`	Retrieve source and background extraction region areas
`get_qvalue`	Returns the statistical significance computed as a q-value
`get_ftest`	Returns the statistical significance computed with the F test
`get_lfactorial`	Returns the natural logarithm of the factorial of the input quantity
`get_source_expr` | `get_bg_expr`	Get the source and background model stack expression
`get_inst_expr` | `get_sinst_expr` | `get_binst_expr`	Get the source and background instrument model stack expression
`get_filter_expr` | `get_bfilter_expr`	Retrieves description of filters applied to source and background datasets
`get_method_expr`	Returns the name of the current optimization method
`get_stat_expr`	Returns the name of the current statistic
`get_defined_models` | `get_defined_inst_models`	Retrives lists of defined source and instrument models
`get_models` | `get_inst_models`	Returns lists of available source and instrument models
`get_model_params` | `get_inst_model_params`	Returns lists of parameter names for source and instrument models
`get_num_par` | `get_num_par_frozen` | `get_num_par_thawed`	Reports the total number of parameters for all defined models, including instrument models
`get_filename` | `get_bfilename` | `get_arf_filename` | `get_rmf_filename`	Retrieve filenames associated with a dataset
`get_dir`	Returns the name of the current directory
`get_verbose`	Returns *Sherpa*'s verbosity

Table 8.1: Summary of *Sherpa*/S-Lang Module `set` Functions

8.6 get_analysis

Module function to retrieve the current analysis setting.

```
String_Type get_analysis([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

The `get_analysis()` function retrieves a string indicating the analysis setting for the specified dataset. The possible return values are “bin” (equivalent to `ANALYSIS CHANNELS`), “keV” (equivalent to `ANALYSIS ENERGY`), “Ang” (equivalent to `ANALYSIS WAVE`), “eV”, “MeV”, “nm”, and “Hz”.

Examples:

1.
```
sherpa> data example.pha
sherpa> instrument = rsp[a](example.rmf,example.arf)
sherpa> get_analysis()
keV
sherpa>
```

8.7 get_arf_axes

Module functions to retrieve the energy/wavelength grid of an ARF associated with source and background data

```
Struct_Type get_arf_axes([Integer_Type])
Struct_Type get_arf_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a “dataspace” is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_arf_axes()` function returns a variable of Struct_Type, which is equivalent to that returned by `get_axes()`. The only difference is that the fields `lo` and `hi` contain data from the `ENERG_LO` and `ENERG_HI` columns from the input ARF file.

One may display the ARF on the same grid using the *Sherpa* plotting command `LPLOT ARF`.

8.8 get_axes

Module functions to get the energy/wavelength/channel grid of source and background datasets.

```
Struct_Type | Array_Type get_axes([Integer_Type])
Struct_Type | Array_Type get_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). The units of the axes are those appropriate for the current *Sherpa* ANALYSIS setting. For instance, if one is working with filtered PHA data in energy-space, what is returned are the low and high bin boundaries in keV (quantities assigned by the EBOUNDS extension of the RMF). (The `get_baxes()` function retrieves the filtered axes for the background associated with the appropriate data set.)

What is returned is an array of structures of length equal to the number of dimensions (*e.g.*, 2 for image data). If the data are one-dimensional, a single Struct_Type variable is returned. There are five structure fields:

- *axistype*: either `Channels`, `Energy`, or `Wavelength`.
- *axisunits*: either `unknown`, `keV`, or Å.
- *lo*: the array of lower bin boundaries; is NULL if there is no bin-width information available.
- *hi*: the array of upper bin boundaries; is NULL if there is no bin-width information available.
- *mid*: NULL if bin-width information is available; otherwise it contains the unbinned axis gridpoints.

One may display data *et al.* on the same grid output by `get_axes` using the *Sherpa* plotting commands `LPLOT DATA` *et al.*

Examples:

1.
```
sherpa> data example.fits
sherpa> foo = get_axes()
sherpa> print(foo[0]) # image pixels, x-axis
   axistype        =  Channels
   axisunits       =  unknown
```

```
   lo              =  Float_Type[512]
   hi              =  Float_Type[512]
   mid             =  NULL
```

2. ```
   sherpa> data example.pha
   sherpa> instrument = rsp[a](example.rmf,example.arf)
   sherpa> foo = get_axes()
   sherpa> print(foo[0]) # energy grid for PHA data
   axistype        =  Energy
   axisunits       =  keV
   lo              =  Float_Type[1024]
   hi              =  Float_Type[1024]
   mid             =  NULL
   ```
3. ```
   sherpa> data example.dat
   sherpa> foo = get_axes()
   sherpa> print(foo[0])  # channel grid for unbinned ASCII data
   axistype        =  Channels
   axisunits       =  unknown
   lo              =  NULL
   hi              =  NULL
   mid             =  Float_Type[100]
   ```

8.9 get_coord

Module function to retrieve the current coordinate setting for 2-D image data.

```
String_Type get_coord([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

The `get_coord()` function retrieves a string indicating the coordinate setting for the specified dataset. For *Sherpa* version 3.0.2, the possible return values are “logical” (equivalent to image), “physical”, or “world” (equivalent to wcs).

Examples:

1. ```
   sherpa> data example.fits
   sherpa> coord 1 physical
   sherpa> get_coord()
   physical
   sherpa>
   ```
   ```

8.10 get_data

Module functions to get the amplitudes of source and background datasets.

```
Array_Type get_data([Integer_Type])
Array_Type get_back([Integer_Type])
Array_Type get_full_data([Integer_Type])
Array_Type get_full_back([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In a typical analysis involving module functions, a user will not seek to manipulate the axes (which are useful primarily for plotting), but to manipulate amplitudes (*i.e.*, $\vec{y}$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$.) A very simple example would be smoothing of data; a more complex example would be the setting of error estimates iteratively in a manner not supported by *Sherpa*. First the user would "get" the error array, then process it, then "set" it back into *Sherpa*.

These functions return arrays containing the y-values of the input datasets. The ones without `full` in the name return arrays comprised of data within the applied filter; those with `full` in the name retrieve the unfiltered datasets.

(If the goal is to use these functions to manipulate data, and to set these data back into *Sherpa*, it is important *not to change the filter in Sherpa* between calling the get and set functions!)

Source and background data may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT DATA` and `LPLOT BACK`.

Examples:

1.
```
sherpa> data example.dat
sherpa> d = get_data(1)
sherpa> printarr(d,2)
59
46
sherpa> d[0] = 80
sherpa> () = set_data(1,d)
sherpa> write data
Write X-Axis: Bin  Y-Axis: Flux (Counts)
          1         80
          2         46
```

8.11 get_dcounts_sum

Calculates the sum of observed counts in source and background datasets using module functions in Sherpa.

```
Struct_Type get_dcounts_sum(Struct_Type)
Struct_Type get_bdcounts_sum(Struct_Type)
Struct_Type get_net_counts_sum(Struct_Type)
Struct_Type get_dcounts_sum([Integer_Type[,{Float_Type | Array_Type}]])
Struct_Type get_bdcounts_sum([Integer_Type[,{Float_Type | Array_Type}]])
Struct_Type get_net_counts_sum([Integer_Type[,{Float_Type | Array_Type}]])
```

Error Return Values: NULL

Arguments:

(1) Structure of form returned by `get_flux_str`; or

(1) Dataset number (default 1)

(2) Evaluation point, or lower-upper bounds (default use all data)

(3) Model component or stack name (default use all appropriate models)

These functions retrieve the summation of observed data counts. `get_net_counts_sum` retrieves the summation of observed *source*, *i.e.*, background-subtracted counts. Note that this function does not require the data to be background-subtracted.

The output of `get_flux_str()`, a structure, can be used as input to `get_dcounts_sum()` and `get_bdcounts_sum()`. One would retrieve this default structure, modify its field values, and pass it to `get_dcounts_sum()` *et al.* See the example below.

Note that numerical arguments are interpreted using *Sherpa*'s current `ANALYSIS` (Section 1.6) setting.

The structure output by these functions contains the following fields:

Field	**Description**
`dataset`	the dataset for which the counts summation is evaluated
`range`	the single point at which the counts is determined, or the range over which the counts are summed; if NULL, the summation is done over the entire dataset range
`comp`	`NULL` for these functions
`value`	the summation of counts
`units`	`NULL` for these functions

Table 8.2: `get_dcounts_sum` Structure Fields

See the related *Sherpa* command `DCOUNTS` for more information.

Examples:

1. Determine the number of counts in a dataset between 2 and 10 keV:

```
sherpa> foo = get_flux_str()
sherpa> print(foo)
dataset          =  1
range            =  NULL
comp             =  NULL
sherpa> foo.range = [2,10]
sherpa> print(get_dcounts_sum(foo).value)
518
sherpa> print(get_net_counts_sum(foo).value)
279.221
```

8.12 get_defined_models

Module functions that return lists of defined *Sherpa* source and instrument models

```
Array_Type get_defined_models()
Array_Type get_defined_inst_models()
```

Error Return Value: NULL

Returns a string array containing the names given by the user to all instantiated *Sherpa* source or instrument models, for use in record-keeping, GUIs, *etc.* (This includes *XSPEC* models and user-defined models.)

See the related *Sherpa* command `SHOW` for more information.

Examples:

1. Define two models; list their names:

```
sherpa> SOURCE = GAUSS[g] + POW[p]
sherpa> mnames = get_defined_models()
sherpa> printarr(mnames)
g
p
```

8.13 get_dir

Module function to retrieve the current path.

```
String_Type get_dir()
```

Error Return Values: NULL

Note that the returned string does *not* include the trailing '/'.

See the related *Sherpa* command `CD` for more information.

8.14 get_eflux

Calculate the energy flux (unconvolved) for source or background datasets using module functions in Sherpa.

```
Struct_Type get_eflux(Struct_Type)
Struct_Type get_beflux(Struct_Type)
Struct_Type get_eflux([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
Struct_Type get_beflux([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
```

Error Return Values: NULL

Arguments:

(1) Structure of form returned by `get_flux_str`; or

(1) Dataset number (default 1)

(2) Evaluation point, or lower-upper bounds (default use all data)

(3) Model component or stack name (default use all appropriate models)

The get_eflux and get_ebflux functions retrieve the energy flux, nominally in units of erg/cm^2/s (if the flux is computed over a range), or erg/cm^2/s/keV (or /Å) (if the flux is computed at a single point). The actual units depend upon the units of the input data and whether or not instrument models have been specified, *etc.*

The output of `get_flux_str()`, a structure, can be used as input to `get_eflux()` and `get_beflux()`. One would retrieve this default structure, modify its field values, and pass it to `get_eflux()` *et al.* See the example below.

Note that numerical arguments are interpreted using *Sherpa*'s current `ANALYSIS` (Section 1.6) setting.

The structure output by these functions contains the following fields:

Field	**Description**
`dataset`	the dataset for which the flux is evaluated
`range`	the single point at which the flux is computed, or the range over which the flux is integrated; if NULL, the integral is done over the entire dataset range

`comp`	the model stack or component for which the flux is computed; if NULL, the whole source/bg stack is used
`value`	the computed flux value
`units`	the flux units

Table 8.3: `get_eflux` Structure Fields

See the related *Sherpa* command `EFLUX` for more information.

Examples:

1. Fit an absorbed power law function to the data set and compute the flux between 2 and 10 keV:

```
sherpa> source= xsphabs[abs]*pow[p1]
sherpa> ignore energy :0.3,10:
sherpa> subtract
sherpa> fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 1401.63
 LVMQT: final statistic value = 235.824 at iteration 5
           abs.nH  0.0626393  10**22 atoms/cm**2
            p1.gamma  1.70739
            p1.ampl  6.81852e-05

sherpa> foo=get_eflux(1,[2,10])
sherpa> print(foo)
dataset         =  1
range           =  Float_Type[2]
comp            =  NULL
value           =  7.72792e-14
units           =  ergs/cm**2/s
sherpa> print(foo.value)
7.72792e-14
```

2. Define a structure foo and use it to compute the flux between 2 and 10 keV:

```
sherpa> foo = get_flux_str()
sherpa> print(foo)
dataset         =  1
range           =  NULL
comp            =  NULL
sherpa> foo.range = [2,10]
sherpa> print(get_eflux(foo).value)
6.76339e-13
sherpa> print(get_eflux(foo).units)
ergs/cm**2/s
sherpa> print(get_eflux(,2.0,"p").value)
1.76965e-13
```

8.15 get_energy_axes

Module functions to retrieve the energy grids of source and background datasets.

```
{Struct_Type | Array_Type} get_energy_axes([Integer_Type])
{Struct_Type | Array_Type} get_energy_baxes([Integer_Type])
{Struct_Type | Array_Type} get_full_energy_axes([Integer_Type])
{Struct_Type | Array_Type} get_full_energy_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_energy_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). Regardless of the current *Sherpa* ANALYSIS setting, this function returns the dataspace in units of energy (`keV`). Otherwise, the function is similar to `get_axes()`.

The function `get_full_energy_axes()` is similar to `get_energy_axes()`, except that `get_full_energy_axes()` returns the original, unfiltered dataspace in units of energy. (And `get_energy_baxes()` and `get_full_energy_baxes()` return filtered and unfiltered dataspaces for the background associated with the source data set, in units of energy.)

One may display data *et al.* on the same grid output by `get_energy_axes` using the *Sherpa* plotting commands `LPLOT DATA` *et al.*

Examples:

1. Read the PHA file with the ARF/RMF defined in the header (qso.pha). Obtain the energy axes information into "foo" and print content of "foo". Print the value of the third energy in "foo". The mid is NULL, so the final print command gives an error information.

```
sherpa> data qso.pha
sherpa> foo=get_energy_axes()
sherpa> print(foo)
axistype        =  Energy
axisunits       =  keV
lo              =  Float_Type[1024]
hi              =  Float_Type[1024]
mid             =  NULL
sherpa> print(foo.lo[2])
0.0292
```

```
sherpa> print(foo.mid[2])
Type Mismatch: Context requires an array.  Scalar not converted
Type Mismatch: print(foo.mid[2]);
```

8.16 get_eqwidth

Module functions that compute the equivalent width of an emission or absorption line in source or background data.

```
Float_Type get_eqwidth([Integer_Type],String_Type,String_Type)
Float_Type get_beqwidth([Integer_Type],String_Type,String_Type)
```

Error Return Values: NULL

Arguments:

(1) Dataset number (default 1)

(2) Expression defining the continuum model

(3) Expression defining the continuum-plus-line model

See the related *Sherpa* command `EQWIDTH` for definitions and more information.

Examples:

1. Model a continuum and emission line complex using a power-law and normalized Gaussian, then compute the equivalent width:

```
sherpa> SOURCE = POW[cont]+NGAUSS[eline]
...
sherpa> FIT
...
sherpa> EQWIDTH 1 (cont,cont+eline)
EW = 0.535073 keV
sherpa> foo = get_eqwidth(1,"cont","cont+eline")
sherpa> print(foo)
0.535073
```

8.17 get_errors

Module functions to get the error estimates of source and background datasets.

```
Array_Type get_errors([Integer_Type])
```

```
Array_Type get_berrors([Integer_Type])
Array_Type get_full_errors([Integer_Type])
Array_Type get_full_berrors([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In a typical analysis involving module functions, a user will not seek to manipulate the axes (which are useful primarily for plotting), but to manipulate amplitudes (*i.e.*, $\vec{y}$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$.) A very simple example would be smoothing of data; a more complex example would be the setting of error estimates iteratively in a manner not supported by *Sherpa*. First the user would "get" the error array, then process it, then "set" it back into *Sherpa*.

These functions return arrays containing the current error estimates associated with input datasets. The ones without `full` in the name return arrays comprised of error estimates within the applied filter; those with `full` in the name retrieve the unfiltered error estimates.

(If the goal is to use these functions to manipulate data, and to set these data back into *Sherpa*, it is important *not to change the filter in Sherpa* between calling the get and set functions!)

Source and background error estimates may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT ERRORS` and `LPLOT BERRORS`.

Examples:

1.
```
sherpa> DATA spec.dat
sherpa> e = get_errors(1)
sherpa> printarr(e,2)
8.72981
7.8374
sherpa> e[0] = 5
sherpa> () = set_errors(1,e)
sherpa> write errors
Write X-Axis: Bin  Y-Axis: Errors
          1          5
          2     7.8374
```

8.18 get_filename

Module functions to retrieve filenames associated with a dataset.

```
String_Type get_filename([Integer_Type])
String_Type get_bfilename([Integer_Type])
```

```
String_Type get_arf_filename([Integer_Type])
String_Type get_rmf_filename([Integer_Type])
```

Error Return Value: NULL

These functions retrieve the source dataset file name, the background dataset file name, the ARF instrument file name, and the RMF instrument file name respectively, for use in record-keeping, GUIs, *etc.*

Note: depending upon how the files are read in, the displayed file name may show the entire directory path. See example below.

See the related *Sherpa* command `SHOW`.

Examples:

1. Input data from files; display file names:

```
sherpa> DATA example.pha
...
sherpa> get_filename
example.pha
sherpa> get_bfilename
/data/simteste/Testing/sherpaTest/data/example2_bkg.pha
sherpa> get_arf_filename
/data/simteste/Testing/sherpaTest/data/example2.arf
sherpa> get_rmf_filename
/data/simteste/Testing/sherpaTest/data/example2.rmf
```

8.19 get_filter

Module functions to get the filter arrays associated with source and background datasets.

```
Array_Type get_filter([Integer_Type])
Array_Type get_bfilter([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

The function `get_filter()` returns an array of type Short_Type that is the same size as the input data set (*i.e.*, the unfiltered data set). The array features two values: 0 for bins outside the currently defined filter, and 1 for bins within the filter. This function may be used in combination with `set_filter` to define new methods of data filtering that go beyond those currently coded in *Sherpa*; *e.g.*, filtering based on data amplitude. (The function `get_bfilter()` returns the filter associated with the background data set.)

Source and background data filters may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT FILTER` and `LPLOT BFILTER`.

Examples:

1.
```
sherpa> data example.dat          # 100 bins, mean amplitude 60
sherpa> foo = get_data()
sherpa> bar = get_filter()
sherpa> bar[where(foo>60)] = 0    # filter out all bins with data > 60
sherpa> () = set_filter(1,bar)
sherpa> write data
Write X-Axis: Bin  Y-Axis: Flux (Counts)
         1         59
         2         46
         3         49
  5         60
         6         60
         8         58
```

2.
```
sherpa> data foo.pha
sherpa> ignore bad
sherpa> notice energy 4:9
sherpa> bar = get_filter_expr()
sherpa> printarr(bar)
ignore source 1 bad
notice source 1 energy 4 : 9
```

8.20 get_filter_expr

Module function to retrieve the strings describing filters applied to source and background datasets.

```
Array_Type get_filter_expr([Integer_Type])
Array_Type get_bfilter_expr([Integer_Type])
```

Error Return Value: NULL

These functions returns an array of filter expressions, for use in record-keeping, GUIs, *etc.* Each string in this array is a command that contributed to the calculation of the filter; the commands were applied to the filter in the order in which they are stored in the array.

See the related *Sherpa* command `SHOW ALL`.

Examples:

1. Retrieve a set of filter expressions applied to dataset 1:

```
sherpa> fltexpr = get_filter_expr()
sherpa> print(fltexpr)
notice source 1 energy 2 : 4
notice source 1 wavelength 12 : 24
```

8.21 get_fit

Module functions to get information about the quality of a fit.

```
Array_Type get_fit([Integer_Type | Array_Type])
Array_Type get_goodness([Integer_Type | Array_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default all data sets), or an array of data set numbers.

The functions `get_fit()` and `get_goodness()` return information about the quality of fit. Currently, the function returns an array of S-Lang variables of Struct_Type. The array is of length 1 if information for only one data set is requested, or $N+1$ if information for N data sets is requested. In the latter case, the total statistic, *etc.*, is given by the first element of the returned array.

The output Struct_Type variable contains the following fields:

- `dataset`: the number of the fit dataset; if NULL, then the Struct_Type variable contains information about all fits.
- `datatype`: one of source, bkgd, or total.
- `stat`: the observed statistic value, or the sum of the statistic values if datatype is set to total.
- `numbins`: the number of bins involved in the fit.
- `dof`: the number of degrees of freedom: `numbins` minus the number of model free parameters.
- `rstat`: the reduced statistic, `stat` divided by `dof`; this field is NULL for the statistics `CASH` and `BAYES`.
- `qval`: the q value (see `get_qvalue()`); this field is NULL for the statistics `CASH` and `BAYES`.

See the related *Sherpa* command `GOODNESS` for more information.

Examples:

1.
```
[...fit two datasets...]
sherpa> res = get_fit()
sherpa> print(res[0])
```

```
dataset          =  NULL    # the total stat, etc., are shown
datatype         =  total   #
stat             =  127.821
numbins          =  200
dof              =  197
rstat            =  0.64884
qval             =  0.999964
sherpa> res = get_res(2)
sherpa> print(res[0])
dataset          =  2       # only the fit info for dataset 2 shown
datatype         =  source
stat             =  63.9103
numbins          =  100
dof              =  98
rstat            =  0.652146
qval             =  0.996969
```

8.21.1 CHANGES IN CIAO 3.2

The get_fit() function now works correctly when given an argument. Prior to CIAO 3.2 it would ignore the supplied argument.

8.22 get_flux2d

Module functions for computing fluxes/summing counts in 2-D images.

```
Struct_Type get_pflux2d([Integer_Type[,String_Type[,String_Type]]])
Struct_Type get_eflux2d([Integer_Type[,String_Type[,String_Type]]])
Struct_Type get_mcounts_sum2d([Integer_Type[,String_Type[,String_Type]]])
Struct_Type get_dcounts_sum2d([Integer_Type[,String_Type]])
```

Error Return Values: NULL

Arguments:

(1) Dataset number (default 1)

(2) Evaluation point, or 2-D region descriptor (default use all data)

(3) Model component or stack name (default use all appropriate models)

These functions retrieve the photon flux, energy flux, summation of convolved predicted model counts, and summation of observed counts respectively. The flux units depend upon the units of the exposure map but may not be correct in the current version of *Sherpa*, so use caution.

The structure output by these functions contains the following fields:

Field	Description
`dataset`	the dataset for which the flux is evaluated/counts summed
`range`	the single point at which the flux is computed/counts summed, or the range over which the flux is integrated/counts summed; if NULL, the integral is done over the entire dataset range
`comp`	the model stack or component for which the flux is computed/counts summed; if NULL, the whole source/bg stack is used
`value`	the computed flux value/sum of counts
`units`	the flux units (`NULL` for counts)

Table 8.4: 2-D Flux Structure Fields

In the current version of the *Sherpa*/S-Lang module, there are no functions for computing fluxes/summations of counts for 2-D background datasets.

Also, in the current version of the *Sherpa*/S-Lang module, there is no way to specify a coordinate system when specifying a 2-D region descriptor; logical coordinates are assumed.

See the related *Sherpa* commands `FLUX`, `EFLUX`, `MCOUNTS`, and `DCOUNTS` for more information.

Examples:

1. Fit a 2-D Gaussian to data without an exposure map; compute photon fluxes and summed model counts (which will be equal in this simplistic example):

```
sherpa> DATA example_img.fits
sherpa> NOTICE FILTER circle(247,257,20)"
sherpa> PARAMPR OFF
sherpa> SOURCE = GAUSS2D[g]
sherpa> FIT
...
sherpa> print(get_pflux2d().value)
3523.46
sherpa> print(get_mcounts_sum2d().value)
3523.46
```

2. Include the exposure map in fitting the data, filter the data, define the source model and fit; obtain the photon flux.

```
sherpa> data img_bin.fits
sherpa> farf2d[em1]
em1.file parameter value ["none"] expmap.fits
em1.norm parameter value [1]

sherpa> instrument=em1
               # include the exposure map in the instrument stack

sherpa> notice filter "rotbox(4022.825,4119.3,15.2,14.95,0)"
```

```
sherpa> foo=get_eflux2d
sherpa> print(foo)
NULL                 # source model has not been defined

sherpa> source=gauss2d[g2]
g2.fwhm parameter value [10.8243]
g2.xpos parameter value [294.5]
g2.ypos parameter value [306.5]
g2.ellip parameter value [0]
g2.theta parameter value [0]
g2.ampl parameter value [30]

sherpa> freeze g2.xpos g2.ypos
sherpa> fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 4.14528e+12
 LVMQT: final statistic value = 2625.34 at iteration 5
            g2.fwhm  1.03876
            g2.ampl  8.5054e-07

sherpa> foo1=get_flux2d

sherpa> print(foo1)
dataset          =  1
range            =  NULL
comp             =  NULL
value            =  1.03989e-06
units            =  photons/cm**2/s
```

8.23 get_flux_str

Retrieves a default structure for use with get_pflux(), *etc.*

```
Struct_Type get_flux_str()
```

The output of `get_flux_str()`, a structure, can be used as input to `get_pflux()` and `get_bpflux()`. One would retrieve this default structure, modify its field values, and pass it to `get_pflux()` *et al.* See the example below.

Examples:

1. Define a structure foo and use it to compute the flux between 2 and 10 keV:

```
sherpa> foo = get_flux_str()
sherpa> print(foo)
```

```
dataset          =  1
range            =  NULL
comp             =  NULL
sherpa> foo.range = [2,10]
sherpa> print(get_flux(foo).value)
0.000166532
sherpa> print(get_flux(foo).units)
photons/cm**2/s
sherpa> print(get_pflux(,[2,10],"p").value)
0.000166532
```

8.24 get_fluxed_spectrum

Retrieve a fluxed spectrum (counts divided by ARF) using module functions in Sherpa.

```
Struct_Type get_fluxed_spectrum([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

This function takes the appropriate data set and divides it by the ARF, if the data are counts (PHA) data and information from an ARF file has also been read in. The function returns a S-Lang variable of Struct_Type containing the counts data divided by the ARF (`data`), the Poisson errors divided by the ARF (`errors`), and (in a future version) the ARF itself, estimated on the counts space energy/wavelength grid (`arf`).

The `data` and `errors` arrays can then be used directly in analyses (after using `set_data()` and `set_errors()`), or can be modified further by the user (*e.g.*, converted from $cts\ cm^{-2}$ to $cts\ cm^{-2}\ s^{-1}$ using `get_exptime()`).

Examples:

1. Obtain the fluxed spectrum and make a plot of the spectrum, include errorbars. Finally write fluxed spectrum to the ascii file.

```
sherpa> foo = get_fluxed_spectrum
sherpa> print(spec1)
data             =  Float_Type[663]
errors           =  Float_Type[663]
arf              =  Float_Type[663]
sherpa> xax=get_energy_axes
                                # obtain the energy scale
sherpa> print(xax)
```

```
axistype        =  Energy
axisunits       =  keV
lo              =  Float_Type[663]
hi              =  Float_Type[663]
mid             =  NULL

sherpa> plot x xax.lo y spec1.data e spec1.errors
                              # this is CHIPS plot command

sherpa> writeascii("fluxed_spec.dat",x.lo, x.hi, spec1.data,
spec1.errors, spec1.arf)

sherpa> $more fluxed_spec.dat
0.3066  0.3212  0.638139       0.172438        32.8455
0.3212  0.3358  0.249041       0.0901943       52.2003
0.3358  0.3504  0.163455       0.0657966       67.2967
0.3504  0.365   0.206631       0.0633644       82.2723
0.365   0.3796  0.213993       0.0594795       93.3986
```

8.25 get_ftest

Module function that returns the statistical significance computed with the F test

`Double_Type get_ftest(Integer_Type, Double_Type, Integer_Type, Double_Type)`

Error Return Value: NULL

Arguments:

(1) number of degrees of freedom d_1

(2) observed statistic value χ_1^2

(3) number of degrees of freedom d_2

(4) observed statistic value χ_2^2

This function returns the tail integral of the F distribution, α_F.

See the related *Sherpa* command `FTEST` for more information.

8.26 get_groups

Module functions for retrieving a grouping or quality array from source and background files.

```
Integer_Type get_groups([Integer_Type])
Integer_Type get_bgroups([Integer_Type])
Integer_Type get_quality([Integer_Type])
Integer_Type get_bquality([Integer_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1).

The `get_(b)groups` and `get_(b)quality` functions allow the user to retrieve grouping and quality information for source and background datasets.

If the file is not grouped or quality is not set, NULL is returned. If the files is grouped or quality defined, an array is returned.

The retrieved array is a `Integer_Type` array of the same length as the input (*grouped*) dataset. A grouping array element set to -1 marks the beginning of a group, while array element set to 1 marks members of that group, so the corresponding bins are treated as one during fitting.

A quality array contains the quality flags for each group: 0 for good (grouped) data; 5 for data labeled as bad by the user (within a tab), and 2 for data labeled as questionable by dmgroup (incomplete groups, *etc.*).

The grouping and quality definitions are based on OGIP standard.

See the related *Sherpa* commands `GROUP` and `QUALITY` for more information.

Examples:

1. Retrive and apply a grouping scheme from the other data set to an ungrouped data set.

```
sherpa> DATA spec.pha
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

.......
sherpa> DATA 2 spec_grp.pha
sherpa> show
....
-----------------
Input data files:
```

```
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231


......

Data 2: spec_grp.pha pha.
Total Size: 131 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231
......
sherpa> g=get_groups(2)
sherpa> print(g)
1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
1
-1
.....
sherpa> set_groups(1,g)
WARNING: any applied filters are being deleted!
1
sherpa> show

Data 1: spec.pi pha.
Total Size: 131 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231
.....
```

In this example, ungrouped and group data are read into *Sherpa*, and then group information is retrieved using **get_groups** from the grouped dataset. A new array **g** is defined whose elements are defined by the grouping of the dataset 2. This grouping scheme is then apply to the ungrouped data set with `set_groups`.

2. Retrieve and apply a grouping scheme from the source data set to ungrouped background data.

```
sherpa> DATA spec.pha
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

 Background 1: bg.pha pha.
  Total Size: 1024 bins (or pixels)
  Dimensions: 1
  Total counts (or values): 662

.......

sherpa> g=get_groups(1)
sherpa> print(g)
1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
1
-1
.....
sherpa> set_bgroups(1,g)
WARNING: any applied filters are being deleted!
1
sherpa> show

Data 1: spec.pi pha.
Total Size: 131 bins (or pixels)
```

```
Dimensions: 1
Total counts (or values): 2231

 Background 1: bg.pha pha.
 Total Size: 131 bins (or pixels)
  Dimensions: 1
  Total counts (or values): 662
......
```

In this example, group data and ungrouped background data are read into *Sherpa*, and then group information is retrieved using `get_groups` from the grouped dataset. A new array `g` is defined whose elements are defined by the grouping of the source data set. This grouping scheme is then apply to the ungrouped background data set with `set_bgroups`.

8.27 get_lfactorial

Module function to compute the natural logarithm of the factorial of the input quantity

```
Double_Type get_lfactorial(Double_Type)
```

This function may be used, *e.g.*, to convert the Cash statistic to the true Poisson log-likelihood in simulation scripts. The input number must be non-negative.

Examples:

1. Compute the natural log of 6 factorial:

```
sherpa> get_lfactorial(6)
6.57925
```

8.28 get_mcounts_sum

Module functions for computing the sum of convolved model counts in source and background datasets.

```
Struct_Type get_mcounts_sum(Struct_Type)
Struct_Type get_bmcounts_sum(Struct_Type)
Struct_Type get_mcounts_sum([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
Struct_Type get_bmcounts_sum([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
```

Error Return Values: NULL

Arguments:

(1) Structure of form returned by `get_flux_str`; or

(1) Dataset number (default 1)

(2) Evaluation point, or lower-upper bounds (default use all data)

(3) Model component or stack name (default use all appropriate models)

These functions retrieve the summation of *convolved* predicted model amplitudes.

The output of `get_flux_str()`, a structure, can be used as input to `get_mcounts_sum()` and `get_bmcounts_sum()`. One would retrieve this default structure, modify its field values, and pass it to `get_mcounts_sum()` *et al.* See the example below.

Note that numerical arguments are interpreted using *Sherpa*'s current `ANALYSIS` (Section 1.6) setting.

The structure output by these functions contains the following fields:

Field	**Description**
`dataset`	the dataset for which the counts summation is evaluated
`range`	the single point at which the counts is determined, or the range over which the counts are summed; if NULL, the summation is done over the entire dataset range
`comp`	the model stack or component for which the counts are summed; if NULL, the whole source/bg stack is used
`value`	the summation of counts
`units`	`NULL` for these functions

Table 8.5: `get_mcounts_sum` Structure Fields

See the related *Sherpa* command `MCOUNTS` for more information.

Examples:

1. Fit a power-law to a dataset; compute the sum of model amplitudes between 2 and 10 keV:

```
sherpa> foo = get_flux_str()
sherpa> print(foo)
dataset         =  1
range           =  NULL
comp            =  NULL
sherpa> foo.range = [2,10]
sherpa> print(get_mcounts_sum(foo).value)
291.483
sherpa> print(get_mcounts_sum(,2.0,"p").value)
12.6129
```

8.29 get_metadata

Module functions to retrieve metadata associated with source and background datasets.

```
Integer_Type get_dimension([Integer])
Float_Type get_exptime([Integer])
Float_Type get_bexptime([Integer])
Float_Type get_backscale([Integer])
Float_Type get_bbackscale([Integer])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

The function `get_dimension()` retrieves the dimensionality of the appropriate data set (*e.g.*, 1 for PHA data, 2 for images). (If no argument is given, the dimensionality of data set 1 is retrieved.)

The function `get_exptime()` retrieves the exposure time of the appropriate data set, in seconds. (The function `get_bexptime()` retrieves the exposure time of the background data set.)

The function `get_backscale()` retrieves the value of the BACKSCAL keyword associated with the appropriate data set (assuming the file from which that data was read was a PHA file). (The function `get_bbackscale()` retrieves the value of the BACKSCAL associated with the appropriate background data set.)

See the related *Sherpa* command `SHOW ALL` for more information.

Examples:

1. ```
   sherpa> data example.pha
   sherpa> get_exptime()
   28678
   ```

8.30 get_method_expr

Module function to retrieve the name of the current optimization method.

```
String_Type get_method_expr()
```

Error Return Value: NULL

This function retrieves the name of the optimization method currently set in *Sherpa*, for use in record-keeping, GUIs, *etc.*
```

See the related *Sherpa* command `SHOW METHOD`.

Examples:

1. Display the *Sherpa* optimization method:

```
sherpa> meth1 = get_method_expr
sherpa> print(meth1)
levenberg-marquardt
sherpa> METHOD SIMPLEX
sherpa> meth2 = get_method_expr
sherpa> print(meth2)
simplex
```

8.31 get_model_params

Access to the default model and instrument parameters of Sherpa from S-Lang.

```
Array_Type get_model_params([String_Type])
Array_Type get_inst_model_params([String_Type])
```

Error Return Value: NULL

Argument:

(1) matching string (default none: list all models)

The two functions allow you to get the default parameter values for all the models in *Sherpa*: `get_model_params()` works on "source" models - such as `xsraymond` and `gauss2d` - whilst `get_inst_model_params()` works on "instrument" models such as `rsp1d`. Each model is described by a structure and the functions return an array of structures, one for each model whose name contains the function argument (if no argument is given all models are returned).

These functions return information about the default parameter values of each model (*i.e.* what the initial values for the models are set to when an instance of the model is created). The `get_par()` function should be used to get the current parameter values of a particular instance of a model.

Field name	Description
`name`	The name of the model.
`type`	The type of the model: "`sherpa`", "`xspec`", or "`usermodel`".
`numpars`	The number of parameters in the model.
`integrated`	What is the default integration mode for this model: 1 for on, 0 for off.
`params`	An array of structures containing the parameter values. This structure is a subset of that returned by the `get_par()` function and is described below.

Table 8.6: Structure returned by `get_model_params()` and `get_inst_model_params()`.

Field name	Description
name	The parameter name.
value	The default parameter value.
min	The default minimum value for the parameter.
max	The default maximum value for the parameter.
units	The units of the parameter. The field will be set to NULL if no units are defined.
frozen	Set to 1 if the parameter is frozen by default, 0 otherwise.

Table 8.7: Contents of the params field.

Examples:

1. List the parameters of the GAUSS1D model:

```
sherpa> gpars = get_model_params("gauss1d")
sherpa> gpars
Struct_Type[1]
sherpa> print(gpars[0])
name             =  gauss1d
type             =  sherpa
numpars          =  3
integrated       =  1
params           =  Struct_Type[3]
```

This shows that the model contains three parameters. The default values for these parameters can be found by examining the array of structures stored in the params field:

```
sherpa> print(gpars[0].params[0])
name             =  fwhm
value            =  10
min              =  2.22507e-308
max              =  1.79769e+308
units            =  NULL
frozen           =  0
sherpa> print(gpars[0].params[1])
name             =  pos
value            =  0
min              =  -1.79769e+308
max              =  1.79769e+308
units            =  NULL
frozen           =  0
sherpa> print(gpars[0].params[2])
name             =  ampl
value            =  1
min              =  -1.79769e+308
max              =  1.79769e+308
units            =  NULL
frozen           =  0
```

2. Find the parameters of all the models whose name contain the word "`raymond`":

```
sherpa> ray = get_model_params("raymond")
sherpa> print(length(ray))
2
sherpa> print(ray[0])
name             =  xsraymond
type             =  xspec
numpars          =  4
integrated       =  1
params           =  Struct_Type[4]
sherpa> print(ray[1])
name             =  xsvraymond
type             =  xspec
numpars          =  15
integrated       =  1
params           =  Struct_Type[15]
```

   The function has returned the default parameter settings for the `XSRAYMOND` and `XSVRAYMOND` models. The type of both models is set to "`xspec`" since they are taken from the *XSPEC* spectral library. As we know the temperature (`kT`) is the first parameter in the list for both models, we can see its default value for the two models by using:

```
sherpa> print(ray[0].params[0])
name             =  kT
value            =  1
min              =  0.008
max              =  64
units            =  keV
frozen           =  0
sherpa> print(ray[1].params[0])
name             =  kT
value            =  6.5
min              =  0.0808
max              =  80
units            =  keV
frozen           =  0
```

3. Find all the instrument models:

```
sherpa> im = get_inst_model_params()
sherpa> print(length(im))
9
sherpa> foreach(im) { s=(); vmessage("%-10s  [%d]",s.name,s.numpars); }
farf        [2]
fexpmap     [2]
fpsf1d      [5]
fpsf2d      [6]
frmf        [1]
rsp         [2]
rsp2d       [7]
```

```
tpsf1d      [5]
tpsf2d      [5]
```

We called `get_inst_model_params()` with no argument so all the instrument models are returned and stored in the variable `im`. The `foreach` command is used to loop through each element of the `im` array, which gets stored into the variable `s`; this is then used in the `vmessage()` function to write out the model name and number of parameters (`s.name` and `s.numpars` respectively).

8.32 get_models

Module functions that return lists of available source and instrument models

```
Array_Type get_models([String_Type])
Array_Type get_inst_models([String_Type])
```

Error Return Value: NULL

Argument:

(1) matching string (default none: get all model names)

Returns a string array containing the names of all appropriate *Sherpa* source or instrument models. (This includes *XSPEC* models and user-defined models.) If a string argument is provided (*e.g.*, "xs"), only those models whose names contain matching substrings will be returned. Beware of case-sensitivity: upper-case matching strings will yield a `NULL` return.

Examples:

1. List all models with `psf` in their names.

```
sherpa> psfmods = get_inst_models("psf")
sherpa> printarr(psfmods)
fpsf1d
fpsf2d
tpsf1d
tpsf2d
```

8.33 get_num_par

Module functions that report the total number of parameters for all defined models, including instrument models

```
Integer_Type get_num_par()
```

```
Integer_Type get_num_par_frozen()
Integer_Type get_num_par_thawed()
```

Error Return Values: NULL

See the related *Sherpa* command `SHOW`.

Examples:

1. Set up two models; determine the number of parameters:

```
sherpa> SOURCE = POW[p] + GAUSS[g]
sherpa> get_num_par
6
sherpa> get_num_par_thawed
5
sherpa> get_num_par_frozen
1
```

8.34 get_par

Module function for getting model parameter values, *etc.*

```
Array_Type get_par([String_Type])
```

Error Return Values: NULL

Argument:

(1) Model name, or model parameter name (default all model parameters)

This function retrieves an array of structures, each of which contains information about a defined model parameter. A typical structure looks as follows:

```
sherpa> GAUSS[g]
sherpa> foo = get_par("g")
sherpa> print(foo[0])
name            =  g.fwhm
model           =  gauss1d
type            =  src
value           =  10
min             =  2.22507e-308
max             =  1.79769e+308
delta           =  -1
units           =  NULL
```

```
frozen          =  0
linked          =  0
linkexpr        =  NULL
```

The fields of the structure are:

Argument	Description
`name`	the name of the parameter
`model`	the model that the parameter belongs to
`type`	source/background-type (`src`) or instrument-type (`inst`) model
`value`	the parameter value (either a number, or a string filename)
`min`	the current (soft) lower bound on allowed parameter values
`max`	the current (soft) upper bound on allowed parameter values
`delta`	the initial step size for the parameter in fits (or -1 to use the default step size)
`units`	parameter units, if known/appropriate
`frozen`	if 1, the parameter value is frozen; if 0, it is thawed
`link`	if 1, the parameter's value is linked to that of other parameter(s); if 0, it is not linked
`linkexpr`	if the parameter is linked to other parameters, the expression showing how it is linked

Table 8.8: `get_par` Structure Fields

See the *Sherpa* command `CREATE` for more information.

Examples:

1. Get a parameter structure; change two fields; set back into *Sherpa*:

```
sherpa> GAUSS[g]
sherpa> foo = get_par("g.pos")
sherpa> print(foo)
name            =  g.pos
model           =  gauss1d
type            =  src
value           =  0
min             =  -3.40282e+38
max             =  3.40282e+38
delta           =  -1
units           =  NULL
frozen          =  0
linked          =  0
linkexpr        =  NULL
sherpa> foo.value = 15.5
sherpa> foo.min = 0.0
sherpa> () = set_par(foo)
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type     Value       Min        Max                 Units
    -----   ----     -----       ---        ---                 -----
```

```
 1   fwhm thawed         10 1.1755e-38 3.4028e+38
 2    pos thawed       15.5          0 3.4028e+38
 3   ampl thawed          1 -3.403e+38 3.4028e+38
```

2. Here we loop through all the defined parameters and display their name and current value:

```
sherpa> erase all
sherpa> paramprompt off
sherpa> xsmekal[gal]
sherpa> xsphabs[abs]
sherpa> ps = get_par()
sherpa> ps
Struct_Type[7]
sherpa> foreach(ps){p=();vmessage("Par %-14s = %g",p.name,p.value);}
Par gal.kT         = 1
Par gal.nH         = 1
Par gal.Abund      = 1
Par gal.Redshift   = 0
Par gal.Switch     = 1
Par gal.norm       = 1
Par abs.nH         = 0.1
```

The initial set of lines are to set up two models ("`gal`", which is an `XSMEKAL` model, and "`abs`", which is an `XSPHABS` model) with default parameter values. The `ps` variable is set here to an array of 7 structures, which we loop through using the S-Lang `foreach` function. Each member of the array is looped through, and stored in the variable `p`. This variable is then used to get the parameters name and value (`p.name` and `p.value` respectively). The whole '`foreach`' statement must be on one line since *Sherpa* does not allow multi-line S-Lang statements. This restriction does not apply to code in a file executed via `evalfile()`.

3. Here we use a small S-Lang function we have written to display parameter values. If the file `print_pars.sl` contains:

```
define print_pars() {
  variable pars = get_par();
  if ( NULL == pars ) {
    print( "No parameter values found!" );
    return;
  }

  vmessage( "# Name                model  frozen  value" );
  foreach ( pars ) {
    variable par = ();
    vmessage( "%-20s  %-10s  %d  %g",
      par.name, par.model, par.frozen, par.value
    );
  }
}
```

then you can say:

```
sherpa> () = evalfile("print_pars.sl")
sherpa> erase all
sherpa> paramprompt off
sherpa> xsmekal[gal]
sherpa> xsphabs[abs]
sherpa> print_pars
# Name               model   frozen  value
gal.kT               xsmekal     0   1
gal.nH               xsmekal     1   1
gal.Abund            xsmekal     1   1
gal.Redshift         xsmekal     1   0
gal.Switch           xsmekal     1   1
gal.norm             xsmekal     0   1
abs.nH               xsphabs     0   0.1
```

although you may find either `list_par()` or `SHOW` more useful.

8.35 get_paramest

Module functions to retrieve the output of parameter estimation methods.

```
Array_Type get_unc()
Array_Type get_proj()
Array_Type get_cov()
Struct_Type get_intunc()
Struct_Type get_intproj()
Struct_Type get_regunc()
Struct_Type get_regproj()
```

See `get_unc`, `get_proj`, `get_cov`, `get_intunc`, `get_intproj`, `get_regunc`, and `get_regproj` for more information.

8.35.1 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields which are described in the documentation for each function.

8.36 get_paramestint

Module functions to retrieve the value and statistic arrays from the most recent run of a parameter estimation method

```
Struct_Type get_intunc()
Struct_Type get_intproj()
```

Error Return Value: NULL

These functions retrieve information from the most recent run of the `INTERVAL-UNCERTAINTY` or `run_intunc`, and `INTERVAL-PROJECTION` or `run_intproj` parameter estimation methods, respectively.

Each returns a structure with five fields:

- `x0`: the grid of parameter values;
- `y`: the statistics as a function of parameter value;
- `name`: the name of the parameter;
- `bfit`: the best-fit value of the parameter; and
- `config`: a Struct_Type variable containing the parameters used to calculate `x0` and `y`.

These functions can be used to retrieve information similar to that provided by the *XSPEC* command `steppar`.

Examples:

1. Fit a dataset; get information about χ^2 as a function of power-law amplitude `p.ampl` after running `INTERVAL-PROJECTION`:

```
sherpa> DATA example.pha
sherpa> SUBTRACT
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POW[p]
sherpa> FIT
...
sherpa> INTERVAL-PROJECTION p.ampl
Interval-Projection: computing grid size with covariance...done.
                     outer grid loop 20% done...
                     outer grid loop 40% done...
                     outer grid loop 60% done...
                     outer grid loop 80% done...
[...plot displayed...]
sherpa> intproj = get_intproj()
sherpa> print(intproj)
x0              =  Float_Type[20]
y               =  Float_Type[20]
name            =  p.ampl
bfit            =  0.000191983
config          =  sherpa_VisParEst_State
sherpa> printarr(intproj.x0,3)
```

```
0.000158651
0.000162159
0.000165668
sherpa> printarr(intproj.y,3)
197.622
195.83
194.238
```

The second-to-last call displays the first three values of the `p.ampl` grid, while the last call displays the best-fit statistic given those `p.ampl` values.

8.36.1 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields - `name`, `bfit`, and `config` - which are described above.

8.37 get_paramestlim

Module functions to determine confidence intervals, and retrieve the parameter bounds.

```
Array_Type get_unc()
Array_Type get_proj()
Array_Type get_cov()
```

Error Return Value: NULL

These functions retrieve information from the most recent run of the `UNCERTAINTY` or `run_unc`, `PROJECTION` or `run_proj`, and `COVARIANCE` or `run_cov` parameter estimation methods, respectively.

Each function returns an array of structures. The length of the array is determined by the number of thawed parameters examined in the most recent run of the parameter estimation method. Each structure has five fields:

- `name`: the name of the parameter;
- `val`: its best-fit value;
- `vlo`: the lower bound for its confidence interval;
- `vhi`: the upper bound for its confidence interval;
- `sigma`: the value of the sigma parameter used to calculate `vlo` and `vhi`.

Examples:

1. Fit a dataset; run covariance and retrieve the results:

```
sherpa> DATA example.pha
sherpa> SUBTRACT
sherpa> PARAMPROMPT OFF
sherpa> SOURCE = POW[p]
sherpa> FIT
...
sherpa> COVARIANCE

Computed for covariance.sigma = 1
        --------------------------------------------------------
        Parameter Name      Best-Fit Lower Bound     Upper Bound
        --------------------------------------------------------
             p.gamma         1.79469  -0.0811155      +0.0811155
             p.ampl      0.000191983  -1.11109e-05    +1.11109e-05

sherpa> cov = get_cov
sherpa> print(length(cov))
2
sherpa> print(cov[0])
name            =  p.gamma
val             =  1.79469
vlo             =  1.71358
vhi             =  1.87581
sigma           =  1
```

2. Repeat the last example, but retrieve the uncertainty for one parameter:

```
sherpa> UNCERTAINTY p.ampl

Computed for uncertainty.sigma = 1
        --------------------------------------------------------
        Parameter Name      Best-Fit Lower Bound     Upper Bound
        --------------------------------------------------------
             p.ampl      0.000191983  -9.38415e-06    +9.4647e-06

sherpa> unc = get_unc()
sherpa> print(unc[0])
name            =  p.ampl
val             =  0.000191983
vlo             =  0.000182599
vhi             =  0.000201448
sigma           =  1
```

8.37.1 CHANGES IN CIAO 3.1

The structures returned by these functions contain an additional field, `sigma`, which is described above.

8.38 get_paramestreg

Module functions to retrieve the value and statistic arrays from the most recent run of a parameter estimation method

```
Struct_Type get_regunc()
Struct_Type get_regproj()
```

Error Return Value: NULL

These functions retrieve information from the most recent run of the `INTERVAL-UNCERTAINTY` or `run_intunc`, and `INTERVAL-PROJECTION` or `run_intproj` parameter estimation methods, respectively.

Each returns a structure with seven fields:

- `x0`: the grid of values for one parameter;
- `x1`: the grid of values for the second parameter;
- `y`: the statistics as a function of parameter values;
- `levels`: an array containing the statistic valuecorresponding to each contour;
- `name`: a two-element array listing the parameter names;
- `bfit`: a two-element array listing the best-fit values of the parameters; and
- `config`: a Struct_Type variable containing the parameters used to calculate `x0`, `x1`, and `y`.

Examples:

1. Fit a dataset; get information about χ^2 as a function of power-law amplitude `p.ampl` and slope `p.gamma`

```
sherpa> DATA spec.dat
sherpa> PARAMPROMPT OFF
sherpa> POLY[p]
sherpa> THAW p.c1 p.c2 p.c3
sherpa> SOURCE = p
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 82.2297
 LVMQT: final statistic value = 62.2247 at iteration 3
            p.c0  61.4774
            p.c1  -0.380228
            p.c2  0.00993229
            p.c3  -7.01741e-05

sherpa> REGPROJ p.c0 p.c1
```

```
Region-Projection: computing grid size with covariance...done.
                   outer grid loop 20% done...
                   outer grid loop 40% done...
                   outer grid loop 60% done...
                   outer grid loop 80% done...
Minimum: 62.2247
Levels are: 64.5207 68.4057 74.0547
[...plot displayed...]
sherpa> regproj = get_regproj()
sherpa> print(regproj)
x0              =  Float_Type[100]
x1              =  Float_Type[100]
y               =  Float_Type[100]
levels          =  Float_Type[3]
name            =  String_Type[2]
bfit            =  Double_Type[2]
config          =  sherpa_VisParEst_State
```

8.38.1 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields - `levels`, `name`, `bfit`, and `config` - which are described above.

8.39 get_pflux

`get_flux()`, `get_pflux()`, and `get_bpflux()` calculate the photon flux (unconvolved) for source and background datasets using the module functions in Sherpa.

```
Struct_Type get_flux(Struct_Type)
Struct_Type get_bflux(Struct_Type)
Struct_Type get_flux([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
Struct_Type get_bflux([Integer_Type[,{Float_Type | Array_Type}[,String_Type]]])
```

Error Return Values: NULL

Arguments:

(1) Structure of form returned by `get_flux_str`; or

(1) Dataset number (default 1)

(2) Evaluation point, or lower-upper bounds (default use all data)

(3) Model component or stack name (default use all appropriate models)

The get_pflux and get_bpflux functions retrieve the photon flux, nominally in units of photons/cm^2/s (if the flux is computed over a range), or photons/cm^2/s/keV (or /Å) (if the flux is computed at a single point). The actual units depend upon the units of the input data and whether or not instrument models (*e.g.* RMF, RSP *etc.*) have been specified, *etc.*

The output of `get_flux_str()`, a structure, can be used as input to `get_pflux()` and `get_bpflux()`. One would retrieve this default structure, modify its field values, and pass it to `get_pflux()` *et al.* See the example below.

Note that numerical arguments are interpreted using *Sherpa*'s current `ANALYSIS` (Section 1.6) setting.

The structure output by these functions contains the following fields:

Field	**Description**
`dataset`	the dataset for which the flux is evaluated
`range`	the single point at which the flux is computed, or the range over which the flux is integrated; if NULL, the integral is done over the entire dataset range
`comp`	the model stack or component for which the flux is computed; if NULL, the whole source/bg stack is used
`value`	the computed flux value
`units`	the flux units

Table 8.9: `get_flux` Structure Fields

See the related *Sherpa* command `FLUX` for more information.

Also see these related *Sherpa*/S-Lang module functions:

Name	**Description**
`get_pflux2d`	Returns photon fluxes in 2-D images
`get_eflux` | `get_beflux`	Returns the unconvolved energy flux for source or background datasets
`get_eflux2d`	Returns energy fluxes in 2-D images
`get_mcounts_sum` | `get_bmcounts_sum`	Returns the sum of convolved model counts in source and background datasets
`get_mcounts_sum2d`	Returns sums of model counts in 2-D images
`get_dcounts_sum` | `get_bdcounts_sum` | `get_net_counts_sum`	Returns the sum of observed counts in source and background datasets
`get_dcounts_sum2d`	Returns sums of observed counts in 2-D images

Table 8.10: Related *Sherpa*/S-Lang Module Functions

Examples:

1. Fit an absorbed power law function to the data set and compute the flux between 2 and 10 keV:

```
sherpa> source= xsphabs[abs]*pow[p1]
```

```
sherpa> fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 1401.63
 LVMQT: final statistic value = 235.824 at iteration 5
           abs.nH  0.0626393  10**22 atoms/cm**2
            p1.gamma  1.70739
            p1.ampl  6.81852e-05

sherpa> foo=get_pflux(1,[2,10])
sherpa> print(foo)
dataset          =  1
range            =  Float_Type[2]
comp             =  NULL
value            =  7.88051e-05
units            =  photons/cm**2/s
sherpa> print(foo.value)
7.88051e-05
```

2. Define a structure foo and use it to compute the flux between 2 and 10 keV:

```
sherpa> foo = get_flux_str()
sherpa> print(foo)
dataset          =  1
range            =  NULL
comp             =  NULL
sherpa> foo.range = [2,10]
sherpa> print(get_flux(foo).value)
0.000166532
sherpa> print(get_flux(foo).units)
photons/cm**2/s
sherpa> print(get_pflux(,[2,10],"p").value)
0.000166532
```

8.40 get_photon_axes

Module functions to retrieve photon-space grids over which models are evaluated.

```
{Struct_Type | Array_Type} get_photon_axes([Integer_Type])
{Struct_Type | Array_Type} get_photon_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_photon_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). The units of the axes are those appropriate for the current *Sherpa* ANALYSIS setting. However, the dataspace is also translated into "photon space". The `get_axes()` function returns the dataspace in "channel space". In essence, "photon space" is what one would see if one could undo the redistribution of energies by the detector. (One can't really "deconcolve" the data in this manner with any confidence, but for the purposes of visualization *only*, the data from the appropriate RMF and ARF files are used to estimate what the data would look like after a "deconvolution".)

Thus, for example, if one is working with filtered PHA data in energy-space, what is returned are the low and high bin boundaries in `keV` (quantities assigned, *e.g.*, by the `ENERG_LO` and `ENERG_HI` columns of the ARF). These boundaries generally do not match the boundaries that would be returned by `get_[b]axes()`.

(The function `get_photon_baxes()` acts as `get_photon_axes()`, save that it retrieves the dataspace for the background associated with the appropriate data set.)

One may display photon-space model amplitudes *et al.* on the same grid output by `get_photon_axes` using the *Sherpa* plotting commands `LPLOT SOURCE` *et al.*

8.41 get_photon_energy_axes

Module functions to retrieve photon-space energy grids over which models are evaluated.

```
{Struct_Type | Array_Type} get_photon_energy_axes([Integer_Type])
{Struct_Type | Array_Type} get_photon_energy_baxes([Integer_Type])
{Struct_Type | Array_Type} get_full_photon_energy_axes([Integer_Type])
{Struct_Type | Array_Type} get_full_photon_energy_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_photon_energy_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). Regardless of the current *Sherpa* ANALYSIS setting, this function returns the dataspace in units of energy (`keV`). However, the dataspace is also translated into "photon space". The `get_energy_axes()` function returns the dataspace in "energy space". In essence, "photon space" is what one would see if one could undo the redistribution of energies by the detector. (One can't really "deconvolve" the data in this manner with any confidence, but for the purposes of visualization *only*, the data from the appropriate RMF and ARF files are used to estimate what the data would look like after a "deconvolution".)

Thus, for example, if one is working with filtered PHA data in energy-space, what is returned are the low and high bin boundaries in `keV` (quantities assigned, *e.g.*, by the `ENERG_LO` and `ENERG_HI` columns of the ARF). But if one is working in wavelength-space or channel-space, the bin boundaries returned are still in `keV`. These boundaries generally do not match the boundaries that would be returned by `get_energy_[b]axes()`.

(The function `get_photon_energy_baxes()` acts as `get_photon_energy_axes()`, save that it retrieves the dataspace for the background associated with the appropriate data set.)

One may display photon-space model amplitudes *et al.* on the same grid output by `get_photon_axes` using the *Sherpa* plotting commands `LPLOT SOURCE` *et al.*

Examples:

1. Read the PHA file with the ARF/RMF defined in the header (qso.pha). Define the model, filter the data and fit.

```
sherpa> data qso.pha
sherpa> foo0=get_photon_energy_axes()
sherpa> print(foo0)
NULL
sherpa> source = xsphabs[abs]*pow[p1]
abs.nH parameter value [0.1]
p1.gamma parameter value [1]
p1.ref parameter value [1]
p1.ampl parameter value [0.00059008]
sherpa>
sherpa> ignore energy :0.2,10:
sherpa> fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 297162
 LVMQT: final statistic value = 275.333 at iteration 7
           abs.nH  0.0245136  10**22 atoms/cm**2
            p1.gamma  1.53707
            p1.ampl  5.91298e-05
```

   Obtain the photon energy axes information into "fooO" and print content of "foo0". Print the value of the 11th low energy in "foo0". The mid is NULL, so the final print command gives an error information.

```
sherpa> foo0=get_photon_energy_axes()
sherpa> print(foo0)
axistype        =  Energy
axisunits       =  keV
lo              =  Float_Type[1077]
hi              =  Float_Type[1077]
mid             =  NULL
sherpa> print(foo0.lo[10])
0.33
```

```
sherpa> print(foo0.mid[10])
Type Mismatch: Context requires an array.  Scalar not converted
Type Mismatch: print(foo0.mid[10]);
```

8.42 get_photon_wave_axes

Module functions to retrieve photon-space wavelength grids over which models are evaluated.

```
{Struct_Type | Array_Type} get_photon_wave_axes([Integer_Type])
{Struct_Type | Array_Type} get_photon_wave_baxes([Integer_Type])
{Struct_Type | Array_Type} get_full_photon_wave_axes([Integer_Type])
{Struct_Type | Array_Type} get_full_photon_wave_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_photon_wave_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). Regardless of the current *Sherpa* ANALYSIS setting, this function returns the dataspace in units of wavelength (Å). However, the dataspace is also translated into "photon space". The `get_wave_axes()` function returns the dataspace in "wavelength space". In essence, "photon space" is what one would see if one could undo the redistribution of energies by the detector. (One can't really "deconvolve" the data in this manner with any confidence, but for the purposes of visualization *only*, the data from the appropriate RMF and ARF files are used to estimate what the data would look like after a "deconvolution".)

Thus, for example, if one is working with filtered PHA data in energy-space, what is returned are the low and high bin boundaries in Å (quantities assigned, *e.g.*, by the `ENERG_LO` and `ENERG_HI` columns of the ARF). But if one is working in energy-space or channel-space, the bin boundaries returned are still in Å. These boundaries generally do not match the boundaries that would be returned by `get_wave_[b]axes()`.

(The function `get_photon_wave_baxes()` acts as `get_photon_wave_axes()`, save that it retrieves the dataspace for the background associated with the appropriate data set.)

One may display photon-space model amplitudes *et al.* on the same grid output by `get_photon_axes` using the *Sherpa* plotting commands `LPLOT SOURCE` *et al.*

8.43 get_qvalue

Module function that returns the statistical significance computed as a q-value.

```
Double_Type get_qvalue(Integer_Type, Double_Type)
```

Error Return Value: NULL

Arguments:

(1) number of degrees of freedom d

(2) observed statistic value χ_o^2

This function returns the tail integral of the χ^2 distribution for the given number of degrees of freedom.

See the related *Sherpa* command `MLR` for more information.

8.44 get_raw_axes

Module functions to retrieve the raw channel number grids of source and background datasets.

```
{Struct_Type | Array_Type} get_raw_axes([Integer_Type])
{Struct_Type | Array_Type} get_raw_baxes([Integer_Type])
{Struct_Type | Array_Type} get_full_raw_axes([Integer_Type])
{Struct_Type | Array_Type} get_full_raw_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_raw_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). Regardless of the current *Sherpa* ANALYSIS setting, this function returns the dataspace in units of channels. Otherwise, the function is similar to `get_axes()`.

The function `get_full_raw_axes()` is similar to `get_raw_axes()`, except that `get_full_raw_axes()` returns the original, unfiltered dataspace in channels. (And `get_raw_baxes()` and `get_full_raw_baxes()` return filtered and unfiltered dataspaces for the background associated with the source data set, in units of channels.)

One may display data *et al.* on the same grid output by `get_raw_axes` using the *Sherpa* plotting commands `LPLOT DATA` *et al.*

8.45 get_record

Module function to return a record of model parameter values at the end of each iteration of the fitting process.

```
Struct_Type get_record()
```

Error Return Value: NULL

The `RECORD` command saves the best-fit model values at the end of each iteration during fits. For instance, if a particular fit with the `POWELL` optimization method took 14 iterations, then 14 values are recorded for each free parameter. Also saved is the statistic value at the end of each iteration. The output of `get_record` is a S-Lang variable of Struct_Type, containing this information.

There are three fields in the output structure:

- `name`: a String_Type array of length M, where M is the number of free model parameters. This array contains the parameter names.
- `val`: a Double_Type array of length $M \times N$, where N is the number of iterations through parameter space during the fitting process. This array records the value of each free model parameter at the end of each step through parameter space.
- `stat`: a Double_Type array of statistic values of length N. This array records the value of the fit statistic at the end of each step through parameter space.

See the related *Sherpa* command `RECORD` for more information.

Examples:

1.
```
sherpa> record
sherpa> fit
...
sherpa> foo = get_record()
sherpa> print(foo)
name            =  String_Type[4]
val             =  Double_Type[4,14]
stat            =  Double_Type[14]
sherpa> variable a = foo.val[0,*]
sherpa> () = curve(a,foo.stat)
```

8.46 get_source

Module functions to retrieve predicted source and background model photon amplitudes.

```
Array_Type get_source([Integer])
Array_Type get_bg([Integer])
Array_Type get_full_source([Integer])
Array_Type get_full_bg([Integer])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

These functions evaluate the source or bg model in *photon* space, and are thus analogous to `get_mcounts()`, *etc.*, which work in *counts* space. What is returned is an array of amplitudes for those photon-space bins which map to the *filtered* data set. (Adding `full` causes unfiltered amplitudes to be returned.) One retrieves the dataspace for these amplitudes using the functions `get_photon_axes()` *et al.*

Filtered source and background model amplitudes in photon-space may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT SOURCE` and `LPLOT BG`.

Examples:

1.
```
sherpa> data example.pha
sherpa> instrument = rsp[rr]("example.rmf", "example.arf")
sherpa> source = xspowlaw[pp]
sherpa> foo = get_source()
```

8.47 get_stackexpr

Module functions to retrieve a model stack expression.

```
String_Type get_source_expr([Integer_Type])
String_Type get_bg_expr([Integer_Type])
String_Type get_inst_expr([Integer_Type])
String_Type get_sinst_expr([Integer_Type])
String_Type get_binst_expr([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) Dataset number (default 1)

These functions get model stack expressions; `get_sinst_expr` and `get_binst_expr` get instrument model stack expressions for the source dataset only, and the background dataset only, respectively.

See the related *Sherpa* commands `SOURCE`, `BACKGROUND`, and `INSTRUMENT` for more information.

Examples:

1. Specify the source model stack expression for dataset 3, then retrieve it:

```
sherpa> PARAMPROMPT OFF
sherpa> GAUSS[g]
sherpa> POW[p]
sherpa> () = set_source_expr(3,"p+g")
sherpa> get_source_expr(3)
(p + g)
```

2. Specify and retrieve a source model stack expression for dataset 1:

```
sherpa> () = set_source_expr("xswabs[xs1]*(pow[p]+gauss[g])")
sherpa> get_source_expr()
(xs1 * (p + g))
```

8.48 get_stat_expr

Module function to retrieve the name of the current statistic.

```
String_Type get_stat_expr()
```

Error Return Value: NULL

This function retrieves the name of the statistic currently set in *Sherpa*, for use in record-keeping, GUIs, *etc.*

See the related *Sherpa* command `SHOW STATISTIC`.

Examples:

1. Display the *Sherpa* statistic setting:

```
sherpa> stat1 = get_stat_expr
sherpa> print(stat1)
chi gehrels
sherpa> STATISTIC CSTAT
sherpa> stat2 = get_stat_expr
sherpa> print(stat2)
cstat
```

8.49 get_statistic

Module functions to get the current value of the statistic comparing source and background data and model values.

```
Double_Type get_statistic([Integer_Type])
Double_Type get_bstatistic([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

The function `get_statistic()` returns the latest evaluated statistic value for the given data set. (The function `get_bstatistic()` does so for the appropriate background data set and background model.)

(Note that this function is not the same as `get_statistics()`, which returns an array showing the value of the fit statistic in each bin. This function returns the sum of the values in the array returned by `get_statistics()`.)

See the related *Sherpa* command `GOODNESS` for more information.

Examples:

1. ```
   sherpa> get_statistic()
   638.148
   ```

8.50 get_stats

Module functions to retrieve predicted model counts, statistics, residuals, sigma residuals, and ratios from source and background datasets.

```
Array_Type get[_full]_[b]mcounts([Integer_Type])
Array_Type get[_full]_[b]statistics([Integer_Type])
Array_Type get[_full]_[b]residuals([Integer_Type])
Array_Type get[_full]_[b]delchi([Integer_Type])
Array_Type get[_full]_[b]ratio([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)
```

The function `get_mcounts()` calls the model associated with the appropriate data set and returns an array containing the predicted model counts amplitudes (*i.e.*, the y-values of the predicted data). The array contains only those y-values which fall within the filter. The other functions all return the result of a comparison of the predicted model counts amplitudes to the data set. Thus, `get_statistics()` returns an array of statistics (χ^2 only), `get_residuals()` returns residuals $(D_i - M_i)$, `get_delchi()` returns residuals divided by sigma ($(D_i - M_i)/\sigma_i$, where σ_i is the estimated error), and `get_ratio()` returns ratios (D_i/M_i).

Arrays containing the whole, unfiltered sets of y-values are returned by using the related functions that have "full" added to their names. Thus, *e.g.*, `get_mcounts()` returns the filtered array of predicted model counts, and `get_full_mcounts()` returns the unfiltered array of predicted model counts.

These functions all use the source model associated with the appropriate data set. (If no argument is given, the default is data set 1.) To instead use the background model and background data set associated with the data set number, append "b" to the function names, *i.e.*, `get_bmcounts()` returns the filtered array of predicted model counts, for the background model, and `get_full_bmcounts()` returns the unfiltered array of predicted model counts, for the background model.

These quantities may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT` [B]`MCOUNTS`, `LPLOT` [B]`STATISTICS`, `LPLOT` [B]`RESIDUALS`, `LPLOT` [B]`DELCHI`, and `LPLOT` [B]`RATION`.

Examples:

1.
```
sherpa> data example.dat
sherpa> source = gauss
sherpa> foo = get_mcounts(1)
sherpa> printarr(foo,2)
0.162616
0.211218
sherpa> write models
Write X-Axis: Bin  Y-Axis: Flux (Counts)
        758     0.1626
        759     0.2112
```

8.51 get_syserrors

Module functions to get the systematic error estimates of source and background datasets.

```
Array_Type get_syserrors([Integer_Type])
Array_Type get_bsyserrors([Integer_Type])
Array_Type get_full_syserrors([Integer_Type])
Array_Type get_full_bsyserrors([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In a typical analysis involving module functions, a user will not seek to manipulate the axes (which are useful primarily for plotting), but to manipulate amplitudes (*i.e.*, $\vec{y}$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$.) A very simple example would be smoothing of data; a more complex example would be the setting of error estimates iteratively in a manner not supported by *Sherpa*. First the user would "get" the error array, then process it, then "set" it back into *Sherpa*.

These functions return arrays containing the current systematic error estimates associated with input datasets. (The quantities are the systematic errors multiplied by the data in each bin, thus one may want to also retrieve the data and divide by them to yield quantities that may be input into `set_syserrors`.) The ones without `full` in the name return arrays comprised of systematic error estimates within the applied filter; those with `full` in the name retrieve the unfiltered systematic error estimates.

(If the goal is to use these functions to manipulate data, and to set these data back into *Sherpa*, it is important *not to change the filter in Sherpa* between calling the get and set functions!)

Source and background systematic errors may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT SYSERRORS` and `LPLOT BSYSERRORS`.

Examples:

1. Retrieve and display systematic errors:

```
sherpa> data example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  /data/simteste/Testing/sherpaTest/data/example2.rmf
ARF is being input from:
  /data/simteste/Testing/sherpaTest/data/example2.arf
Background data are being input from:
  /data/simteste/Testing/sherpaTest/data/example2_bkg.pha
sherpa> syserr = get_full_syserrors()
sherpa> printarr(syserr,3)
0
0
0
```

8.52 get_verbose

Module function to display *Sherpa*'s current verbosity setting.

```
Void_Type get_verbose()
get_verbose
```

This function retrieve's *Sherpa*'s verbosity setting.

The current verbosity settings are:

Setting	Displays:
0	(almost) no output (useful for scripts/batch jobs)
1	normal output; many warnings are suppressed and the results of each fit iteration are not shown (the default)
2	full output

Table 8.11: Verbosity Settings for `get_verbose`

8.53 get_wave_axes

Module functions to retrieve the wavelength grids of source and background datasets.

```
{Struct_Type | Array_Type} get_wave_axes([Integer_Type])
{Struct_Type | Array_Type} get_wave_baxes([Integer_Type])
{Struct_Type | Array_Type} get_full_wave_axes([Integer_Type])
{Struct_Type | Array_Type} get_full_wave_baxes([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In *Sherpa* parlance, a "dataspace" is an N-dimensional grid defined by the independent variables of the dataset (*i.e.*, $\vec{x}_i$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$). Simple examples include the `CHANNELS` array in PHA datasets and the pixel numbers along each axis of FITS images.

The `get_wave_axes()` function retrieves the dataspace, or filtered data set axes of the appropriate data set (if no argument is given, the axes for data set 1 are retrieved). Regardless of the current *Sherpa* ANALYSIS setting, this function returns the dataspace in units of wavelength. Otherwise, the function is similar to `get_axes()`.

The function `get_full_wave_axes()` is similar to `get_wave_axes()`, except that `get_full_wave_axes()` returns the original, unfiltered dataspace in units of wavelength. (And `get_wave_baxes()` and `get_full_wave_baxes()` return filtered and unfiltered dataspaces for the background associated with the source data set, in units of wavelength.)

One may display data *et al.* on the same grid output by `get_wave_axes` using the *Sherpa* plotting commands `LPLOT DATA` *et al.*

8.54 get_weights

Module functions to get the statistical weights of source and background datasets.

```
Array_Type get_weights([Integer_Type])
Array_Type get_bweights([Integer_Type])
Array_Type get_full_weights([Integer_Type])
Array_Type get_full_bweights([Integer_Type])
```

Error Return Value: NULL

Arguments:

(1) data set number (default 1)

In a typical analysis involving module functions, a user will not seek to manipulate the axes (which are useful primarily for plotting), but to manipulate amplitudes (*i.e.*, $\vec{y}$ in the expression $\vec{y} = f(\vec{x}_0, \vec{x}_1, ..., \vec{x}_{N-1})$.) A very simple example would be smoothing of data; a more complex example would be the setting of error estimates iteratively in a manner not supported by *Sherpa*. First the user would "get" the error array, then process it, then "set" it back into *Sherpa*.

These functions return arrays containing the current statistical weights associated with input datasets. The ones without `full` in the name return arrays comprised of statistical weights within the applied filter; those with `full` in the name retrieve the unfiltered statistical weights.

(If the goal is to use these functions to manipulate data, and to set these data back into *Sherpa*, it is important *not to change the filter in Sherpa* between calling the get and set functions!)

Source and background statistical weights may be displayed, *e.g.*, via the *Sherpa* plotting commands `LPLOT WEIGHTS` and `LPLOT BWEIGHTS`.

Examples:

1.
```
sherpa> DATA spec.dat
sherpa> w = get_weights(1)
sherpa> printarr(w,2)
1
1
sherpa> e[0] = 0.5
sherpa> () = set_weights(1,e)
sherpa> write weights
Write X-Axis: Bin  Y-Axis: Errors
          1        0.5
          2          1
```

8.55 is

Summary of *Sherpa*/S-Lang module query functions.

The `is` functions of the *Sherpa*/S-Lang module answer questions, returning 1 for yes/true and 0 for no/false, and, if appropriate, NULL for inappropriate queries (such as querying about background subtraction for a non-existant dataset).

Name	Description
`is_thawed`	Determines if the specified parameter(s) are thawed
`is_frozen`	Determines if the specified parameter(s) are frozen
`is_subtracted`	Determines if a specified dataset is background-subtracted

Table 8.12: Summary of *Sherpa*/S-Lang Module `is` Functions

8.56 is_paramset

Module functions to determine if whether the specified parameter(s) are thawed or frozen.

```
Integer_Type is_thawed([String_Type])
Integer_Type is_frozen([String_Type])
```

Arguments:

(1) Model name, or model parameter name (default all model parameters)

If:

- no argument is provided, these functions will return 1 if *all* model parameters are thawed, or frozen;
- a model component name is provided, these functions return 1 if all parameters of that model are thawed, or frozen; and
- if a model parameter name is provided, these functions return 1 if that parameter is thawed, or frozen.

See the related *Sherpa* commands `THAW` and `FREEZE` for more information.

Examples:

1. Determine if all parameters of a specified Gaussian model are thawed, then determine if the `ampl` parameter is frozen:

   ```
   sherpa> PARAMPROMPT OFF
   sherpa> GAUSS[g]
   sherpa> is_thawed("g")
   1
   sherpa> is_frozen("g.ampl")
   0
   ```

   The 1 returned from the first call indicates that yes, all parameters of the Gaussian model named `g` are thawed. The 0 returned from the second call indicates that no, the parameter g.ampl is *not* frozen.

8.57 is_subtracted

Module function to determine if whether the specified dataset is background-subtracted.

```
Integer_Type is_subtracted([Integer_Type])
```

Arguments:

(1) Dataset number (default 1)

This function returns 1 if the background has been subtracted in the specified source dataset.

See the related *Sherpa* commands `SUBTRACT` and `UNSUBTRACT` for more information.

Examples:

1. Determine if dataset 1 is background subtracted:

   ```
   sherpa> DATA example.pha
   The inferred file type is PHA.  If this is not what you want, please
   specify the type explicitly in the data command.
   WARNING: using systematic errors specified in the PHA file.
   RMF is being input from:
     <directory_path>/example2.rmf
   ARF is being input from:
     <directory_path>/example2.arf
   Background data are being input from:
     <directory_path>/example2_bkg.pha
   sherpa> is_subtracted
   0
   sherpa> SUBTRACT
   sherpa> is_subtracted(1)
   1
   ```

   The 0 returned from the first call indicates that no, dataset 1 is not background subtracted. (Unlike *XSPEC*, *Sherpa* does not automatically subtract background data upon input.) The 1 returned from the second call, issued after the `SUBTRACT` command, indicates that yes, dataset 1 is now background-subtracted.

8.58 list_par

Module function that lists information about all defined model parameters.

```
Void_Type list_par()
list_par
```

This function causes a table to be displayed with the following columns:

Label	**Displays:**
`Name`	The parameter name in the form <**`modelname`**>**`.`**<**`paramname`**>
`Type`	Source/background model (**`src`**) or instrument model (**`inst`**)
`Value`	The current value of the parameter, or the filename if appropriate
`Lnk`	If 1, indicates that the listed parameter's value is linked to other parameter(s) (which are listed)
`Frz`	If 1, indicates that the parameter's value is frozen
{`Min` | `Max`}	The current (soft, *i.e.*, changable) parameter lower and upper bounds
`Delta`	The initial step size for the parameter in fits, with a negative value indicating that a default (usually 0.01 times **`Value`**) will be used

Table 8.13: Columns in `list_par` Output

See the related *Sherpa* command `SHOW` for more information.

Examples:

1. Set up a *Sherpa* session; use `list_par` to show parameters:

```
sherpa> SOURCE = GAUSS[g]
sherpa> GAUSS[g2]
sherpa> g2.pos => g.pos
sherpa> RSP[a](example.rmf,example.arf)
sherpa> list_par
  #    Name Type       Value Lnk Frz         Min         Max       Delta
  1  g.fwhm  src      2.6593   0   0    0.026593      265.93          -1
  2   g.pos  src     0.94418   0   0    0.027614      14.549          -1
  3  g.ampl  src  0.00010564   0   0  1.0564e-06    0.010564          -1
  4 g2.fwhm  src      3.5711   0   0    0.035711      357.11          -1
  5  g2.pos  src     0.94418   1 => g.pos
  6 g2.ampl  src   0.0010453   0   0  1.0453e-05     0.10453          -1
  7   a.rmf inst  "example.rmf"
  8   a.arf inst  "example.arf"
```

8.59 list_paramest

Module functions listing the current and default values of the parameters used to configure each *Sherpa* parameter estimation method.

```
list_unc
list_proj
list_cov
list_intunc
```

```
list_intproj
list_regunc
list_regproj
```

These functions cause tables to be displayed that show the names, current values, default values, and descriptions of the *Sherpa* configuration variables (also called "state objects") `sherpa.unc` *et al.* The variable values may be modified at the command line (see the example below).

To reset the current values to default values, use the functions `restore_unc` *et al.*

These functions replace the *Sherpa* commands `SHOW UNCERTAINTY` *et al.* See the related *Sherpa* commands `UNCERTAINTY`, `PROJECTION`, `COVARIANCE`, `INTERVAL-UNCERTAINTY`, `INTERVAL-PROJECTION`, `REGION-UNCERTAINTY`, and `REGION-PROJECTION` for more information.

Examples:

1. Modify the *Sherpa* state variable `sherpa.regproj`; display current values with `list_regproj`:

```
sherpa> sherpa.regproj.nloop = [30,20]
sherpa> sherpa.regproj.sigma = [1.6,2.6]  # 90 and 95 percent contours
sherpa> list_regproj
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            1               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [0,0]           [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop       [30,20]         [10,10]   Number of grid points, each axis
sigma     [1.6,2.6]         [1,2,3]      Number of sigma, each contour
```

8.60 load

Summary of *Sherpa*/S-Lang module functions that load data into *Sherpa*

The `load` functions of the *Sherpa*/S-Lang module are functional analogues to the *Sherpa* command `READ`, with a key difference: `READ` simply inputs file-based data into *Sherpa*, while the `load` functions may be invoked after data are manipulated at the command-line. A simple intuitive example is the processing of read-in time-series data before analysis, by shifting the photon arrival times by some constant (*e.g.*, to make the arrival time of the first photon $t_{\rm new} = 0$).

Name	**Description**
`load_dataset`	Loads a source dataset from a file or a S-Lang structure
`load_backset`	Loads a background dataset from a file or a S-Lang structure

{`load_ascii` | `load_bascii`}	Loads a source or background dataset from an ASCII file
{`load_fitsbin` | `load_bfitsbin`}	Loads a source or background dataset from a FITS binary table file
{`load_pha` | `load_bpha`}	Loads a source or background dataset from a PHA-format file
`load_pha2`	Loads source datasets from a PHA Type II-format file
{`load_image` | `load_bimage`}	Loads a source or background dataset from a FITS image file
`load_back_from`	Loads the contents of a background file named in the header of a PHA data file
`load_arf`	Loads the contents of an ARF file/structure, placing them in an RSP instrument model
`load_rmf`	Loads the contents of an RMF file/structure, placing them in an RSP instrument model
`load_inst`	Loads the contents of an ARF and RMF file/structure, placing them in an RSP instrument model
{`load_inst_from` | `load_binst_from`}	Loads the contents of ARF and RMF files named in the header of a PHA data file, placing them in an RSP instrument model

Table 8.14: Summary of *Sherpa*/S-Lang Module `load` Functions

8.61 load_arf

Module function to load data from an ARF file into *Sherpa*

```
Integer_Type load_arf(String_Type, {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Name of instrument (RSP) model instance

(2) ARF filename, or

(2) S-Lang variable output by `readarf()`

This function loads data from an ARF file into *Sherpa*, assigning the data to the `RSP` model instance identified by the first argument. If no such model exists before the call, a new `RSP` model is created.

See the related *Sherpa* models `RSP` and `FARF` for more information.

Examples:

1. `sherpa> () = load_arf("rr","example.arf")`

   In this example, the data from the ARF file example.arf is assigned to the `RSP` model "rr".

2. ```
   sherpa> arfst = readarf("example.arf")
   sherpa> () = load_arf("rr","arfst")
   sherpa> () = sherpa_eval("instrument 1 = rr")
   ```

   Here the ARF data are read into the S-Lang variable "arfst". The `load_arf()` function then reads the data from arrays which are part of the variable "arfst".

8.62 load_ascii

Module function to load ASCII data into *Sherpa*

```
Integer_Type load_ascii([Integer_Type], {String_Type | Struct_Type})
Integer_Type load_bascii([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) ASCII filename, or

(2) S-Lang variable of Struct_Type (*e.g.*, type of variable returned by VARMM `readascii()`)

The function `load_ascii()` loads data from an ASCII file into a Sherpa data set. The function `load_bascii()` loads data from an ASCII file into the background associated with a Sherpa source data set. (The brackets indicate an optional letter in the function name, with "[b]" representing functions to use with background data. Note that background functions currently assume that the setting of `multiback` in the *Sherpa* state object is 0, *i.e.*, that only one background dataset is allowed per source dataset.)

Currently, users may only input 1D data (*i.e.*, two columns, with the first assumed to contain values of the independent variable x and the second to contain values of the dependent variable $y = f(x)$). Field names in the struct do not matter, although in the current implementation an `_nrows` field is needed to establish the length of the arrays.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_ascii(,"example.dat")
   sherpa> () = load_ascii(1,"example.dat")
   ```

```
sherpa> () = load_bascii(,"example.dat")
sherpa> () = load_bascii(1,"example.dat")
```

The first two commands will load data from the ASCII file into Sherpa source data set 1. The last two commands will load data from the ASCII file into the background associated with Sherpa source data set 1.

2. ```
   sherpa> foo = readascii("example.dat")
   sherpa> () = load_ascii(1,foo)
   ```

   Here the data are read into the S-Lang variable "foo". The `load_ascii()` function then reads the data from arrays which are part of the variable "foo".

8.63 load_back_from

Module function to load background from a PHA file into *Sherpa*

```
Integer_Type load_back_from([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) PHA filename, or

(2) S-Lang variable output by `readpha()`

The function `load_back_from()` loads the contents of a background PHA file into *Sherpa*. The file name of the background file is determined from the header keywords of the input (source) PHA file or S-Lang variable.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_back_from(,"example.pha")
   sherpa> () = load_back_from(1,"example.pha")
   ```

   These two commands will load the background data from the file background file listed in the header of example.pha.

2. ```
   sherpa> foo = readpha("example.pha")
   sherpa> () = load_back_from(1,foo)
   ```

   The name of the background file is one of the fields of the "foo" variable. The background data, and only the background data, is then loaded with the `load_back_from` function.
   ```

8.64 load_backset

Module function to load background data into *Sherpa*

```
Integer_Type load_backset([Integer_Type], {String_Type | Struct_Type | Array_Type}, [,Array_Type, ...])
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) File name, or

(2) S-Lang variable of Struct_Type (*e.g.*, type of variable returned by VARMM `readfile()`), or

(2) S-Lang variable of Array_Type

(3) More S-Lang variables of Array_Type

This function acts as a wrapper around the filetype-specific load functions (*e.g.*, `load_bascii`, `load_bpha`, *etc.*). If a file name is provided, then the data are read in via the VARMM library `readfile()` function. If a S-Lang variable of Struct_Type is provided, then the data are loaded from the fields of that variable. If one or more S-Lang variables of Array_Type are provided, then the data are passed to the functions `set_axes()` and `set_data()`. A maximum of 8 arrays, for for data of up to 7 dimensions, can be passed as arguments of Array_Type.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_backset(,"example.dat")
   sherpa> () = load_backset(1,"example.pha")
   ```

   These two commands are equivalent; both load data from file into the background associated with the first source data set. In the first case, data are loaded from an ASCII file, and in the second, data are loaded from a PHA file; `load_backset()` calls the appropriate function to read from ASCII and FITS files respectively.

2. ```
   sherpa> foo = readfile("example.pha")
   sherpa> () = load_backset(1,foo)
   sherpa> () = load_backset(1,foo.channels, foo.counts)
   ```

   Here the data are loaded into a S-Lang variable of Struct_Type. In the first call to `load_backset()`, this S-Lang variable is passed to the function. In the second call, two arrays are passed to `load_backset()`. The data in “foo.channels” and “foo.counts” are used as the x and y arrays respectively; but the other fields of the variable “foo” are not used. This means the second call to `load_backset()` is not quite the equivalent of the first.

8.65 load_dataset

Module function to load data into *Sherpa*

```
Integer_Type load_dataset([Integer_Type], {String_Type | Struct_Type | Array_Type}, [,Array_Type, ...])
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) File name, or

(2) S-Lang variable of Struct_Type (*e.g.*, type of variable returned by VARMM `readfile()`), or

(2) S-Lang variable of Array_Type

(3) More S-Lang variables of Array_Type

This function acts as a wrapper around the filetype-specific load functions (*e.g.*, `load_ascii`, `load_pha`, *etc.*). If a file name is provided, then the data are read in via the VARMM library `readfile()` function. If a S-Lang variable of Struct_Type is provided, then the data are loaded from the fields of that variable. If one or more S-Lang variables of Array_Type are provided, then the data are passed to the functions `set_axes()` and `set_data()`. A maximum of 8 arrays, for for data of up to 7 dimensions, can be passed as arguments of Array_Type.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_dataset(,"example.dat")
   sherpa> () = load_dataset(1,"example.pha")
   ```

   These two commands are equivalent; both load data from file into data set 1. In the first case, data are loaded from an ASCII file, and in the second, data are loaded from a PHA file; `load_dataset()` calls the appropriate function to read from ASCII and FITS files respectively.

2. ```
   sherpa> foo = readfile("example.pha")
   sherpa> () = load_dataset(1,foo)
   sherpa> () = load_dataset(1,foo.channels, foo.counts)
   ```

   Here the data are loaded into a S-Lang variable of Struct_Type. In the first call to `load_dataset()`, this S-Lang variable is passed to the function. In the second call, two arrays are passed to `load_dataset()`. The data in "foo.channels" and "foo.counts" are used as the x and y arrays respectively; but the other fields of the variable "foo" are not used. This means the second call to `load_dataset()` is not quite the equivalent of the first.

8.66 load_fitsbin

Module functions to load source and background data from a FITS binary table into *Sherpa*

```
Integer_Type load_fitsbin([Integer_Type], {String_Type | Struct_Type})
Integer_Type load_bfitsbin([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) FITS binary table filename, or

(2) S-Lang variable output by `readbintab()`

The function `load_fitsbin()` loads data from a FITS binary table into a Sherpa data set. The function `load_bfitsbin()` loads data from a FITS binary table into the background associated with a Sherpa source data set. (The brackets indicate an optional letter in the function name, with "[b]" representing functions to use with background data. Note that background functions currently assume that the setting of `multiback` in the *Sherpa* state object is 0, *i.e.*, that only one background dataset is allowed per source dataset.)

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_fitsbin(,"example.fits")
   sherpa> () = load_fitsbin(1,"example.fits")
   sherpa> () = load_bfitsbin(,"example.fits")
   sherpa> () = load_bfitsbin(1,"example.fits")
   ```

   The first two commands will load data from the FITS binary table into Sherpa source data set 1. The last two commands will load data from the FITS binary table into the background associated with Sherpa source data set 1.

2. ```
   sherpa> foo = readbintab("example.dat")
   sherpa> () = load_fitsbin(1,foo)
   ```

   Here the data are read into the S-Lang variable "foo". The `load_fitsbin()` function then reads the data from arrays which are part of the variable "foo".

3. ```
   sherpa gt; () = load_fitsbin(1,"example.pha[cols CHANNEL,COUNTS]")
   ```

   Here is an example using Data Model syntax. With this function, the file name can include virtual file syntax to specify the FITS file extension from which data are read (the default is the first FITS binary table extension), and the columns from which data are read.
   ```

8.67 load_image

Module functions to load source and background data from a FITS image into *Sherpa*

```
Integer_Type load_image([Integer_Type], {String_Type | Struct_Type})
Integer_Type load_bimage([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) FITS image filename, or

(2) S-Lang variable output by `readimage()`

The function `load_image()` loads data from a FITS image file into a Sherpa data set. The function `load_bimage()` loads data from a FITS image file into the background associated with a Sherpa source data set. (The brackets indicate an optional letter in the function name, with "[b]" representing functions to use with background data. Note that background functions currently assume that the setting of `multiback` in the *Sherpa* state object is 0, *i.e.*, that only one background dataset is allowed per source dataset.) WCS information is automatically loaded from the header into *Sherpa*, if it exists.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_image(,"example.fits")
   sherpa> () = load_image(1,"example.fits")
   sherpa> () = load_bpha(,"example.fits")
   sherpa> () = load_bpha(1,"example.fits")
   ```

   The first two commands will load data from the FITS image file into Sherpa source data set 1. The last two commands will load data from the FITS image file into the background associated with Sherpa source data set 1.

2. ```
   sherpa> foo = readimage("example.pha")
   sherpa> () = load_image(1,foo)
   ```

   Here the data are read into the S-Lang variable "foo". The `load_image()` function then reads the data from arrays which are part of the variable "foo".

8.68 load_inst

Module function to load data from ARF and RMF files into *Sherpa*. `load_instrument` is an alternate name.

```
Integer_Type load_inst({String_Type | Struct_Type},{String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) ARF filename, or

(1) S-Lang variable output by `readarf()`

(2) RMF filename, or

(2) S-Lang variable output by `readrmf()`

This function loads data from an ARF file and an RMF file into *Sherpa*, assigning them to an automatically named `RSP` model instance.

Note that the order (ARF,RMF; RMF,ARF) does not actually matter, except that currently an unsuppressible library error message appears when the RMF is listed first. This error message may be safely ignored.

See the related *Sherpa* models `RSP`, `FRMF`, and `FARF` for more information.

Examples:

1. ```
   sherpa> rmfst = readrmf("example.rmf")
   sherpa> () = load_inst("example.arf",rmfst)
   ```
   In this example, RMF data are read from example.rmf and put into the S-Lang variable "rmfst"; this variable is then passed as an argument to `load_inst()`. The ARF data are read directly from the example.arf file.

8.69 load_inst_from

Module function to load data from ARF and RMF files into *Sherpa* source or background instrument models.

```
Integer_Type load_inst_from({String_Type | Struct_Type})
Integer_Type load_binst_from({String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) PHA filename, or

(1) S-Lang variable output by `readpha()`
```

This function loads data from an ARF file and an RMF file into *Sherpa*, assigning them to an automatically named `RSP` model instance. The file names are determined from the header keywords of the input PHA file or S-Lang variable.

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_inst_from("example.pha")
   sherpa> phast = readpha("example.pha")
   sherpa> () = load_inst_from(phast)
   ```
   In the first example, the first call causes ARF and RMF files to be read from the files named in the header of the PHA file. In the second example, the data from the PHA file is first read into a S-lang variable; that S-Lang variable is then passed as an argument to the `load_inst_from()` function.

8.70 load_pha

Module function to load source and background data from PHA Type I files into *Sherpa*

```
Integer_Type load_pha([Integer_Type], {String_Type | Struct_Type})
Integer_Type load_bpha([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) PHA Type I filename, or

(2) S-Lang variable output by `readpha()`

The function `load_pha()` loads data from a PHA Type I file into a Sherpa data set. The function `load_bpha()` loads data from a PHA Type I file into the background associated with a Sherpa source data set. (The brackets indicate an optional letter in the function name, with "[b]" representing functions to use with background data. Note that background functions currently assume that the setting of `multiback` in the *Sherpa* state object is 0, *i.e.*, that only one background dataset is allowed per source dataset.)

If the file contains the header keywords `BACKFILE`, `RESPFILE`, and/or `ANCRFILE`, the background, RMF, and/or ARF file contents are automatically read in as well. (`BACKFILE` is ignored for `load_bpha`.)

See the related *Sherpa* command `READ` for more information.

Examples:

1. ```
   sherpa> () = load_pha(,"example.pha")
   ```

```
sherpa> () = load_pha(1,"example.pha")
sherpa> () = load_bpha(,"example.pha")
sherpa> () = load_bpha(1,"example.pha")
```

The first two commands will load data from the PHA Type I file into Sherpa source data set 1. The last two commands will load data from the PHA Type I file into the background associated with Sherpa source data set 1.

2.
```
sherpa> foo = readpha("example.pha")
sherpa> () = load_pha(1,foo)
```

Here the data are read into the S-Lang variable "foo". The `load_pha()` function then reads the data from arrays which are part of the variable "foo".

8.71 load_pha2

Module function to load source data from PHA Type II files into *Sherpa*

```
Integer_Type load_pha2([Integer_Type], {String_Type | Struct_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) Data set number (default 1)

(2) PHA Type II filename, or

(2) S-Lang variable output by `readpha()`

The function `load_pha2()` loads data from a PHA Type II file into a Sherpa data set. If the file contains the header keywords `BACKFILE`, `RESPFILE`, and/or `ANCRFILE`, the background, RMF, and/or ARF file contents are automatically read in as well. If the file contains the columns `BACKGROUND_UP` and `BACKGROUND_DOWN`, their contents are combined into a single background dataset, which is then read in.

Note that there is no equivalent function currently for inputting background data from a PHA Type II file.

See the related *Sherpa* command `READ` for more information.

Examples:

1.
```
sherpa> () = load_pha2(,"example.pha")
sherpa> () = load_pha2(1,"example.pha")
```

These two commands will load data from the PHA Type II file into Sherpa source data sets, starting with data set 1.

2.
```
sherpa> foo = readpha("example.pha")
sherpa> () = load_pha2(1,foo)
```

Here the data are read into the S-Lang variable “foo”. The `load_pha2()` function then reads the data from arrays which are part of the variable “foo”.

8.72 load_rmf

Module function to load data from an RMF file into *Sherpa*

```
Integer_Type load_rmf(String_Type, String_Type | Struct_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Name of instrument (RSP) model instance

(2) RMF filename, or

(2) S-Lang variable output by `readrmf()`

This function loads data from an RMF file into *Sherpa*, assigning the data to the `RSP` model instance identified by the first argument. If no such model exists before the call, a new `RSP` model is created.

See the related *Sherpa* models `RSP` and `FRMF` for more information.

Examples:

1. `sherpa> () = load_rmf("rr","example.rmf")`

   In this example, the data from the RMF file example.rmf is assigned to the `RSP` model “rr”.

2. ```
   sherpa> rmfst = readrmf("example.arf")
   sherpa> () = load_rmf("rr","rmfst")
   sherpa> () = sherpa_eval("instrument 1 = rr")
   ```

   Here the RMF data are read into the S-Lang variable “rmfst”. The `load_rmf()` function then reads the data from arrays which are part of the variable “rmfst”.

8.73 restore_paramest

Module functions to restore the default values of the parameters used to configure each *Sherpa* parameter estimation method.

```
restore_unc
restore_proj
restore_cov
```
```

```
restore_intunc
restore_intproj
restore_regunc
restore_regproj
```

These functions restore the default values of the *Sherpa* configuration variables (also called "state objects") `sherpa.unc` *et al.*

To display the current values, use the functions `list_unc` *et al.*

See the related *Sherpa* commands `UNCERTAINTY`, `PROJECTION`, `COVARIANCE`, `INTERVAL-UNCERTAINTY`, `INTERVAL-PROJECTION`, `REGION-UNCERTAINTY`, and `REGION-PROJECTION` for more information.

Examples:

1. Modify the *Sherpa* state variable `sherpa.regproj`; display current values with `list_regproj`; restore the default values:

```
sherpa> sherpa.regproj.nloop = [30,20]
sherpa> sherpa.regproj.sigma = [1.6,2.6]  # 90 and 95 percent contours
sherpa> list_regproj
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            1               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [0,0]           [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop       [30,20]         [10,10]   Number of grid points, each axis
sigma     [1.6,2.6]         [1,2,3]      Number of sigma, each contour
sherpa> restore_regproj
sherpa> list_regproj
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            1               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [0,0]           [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop       [10,10]         [10,10]   Number of grid points, each axis
sigma       [1,2,3]         [1,2,3]      Number of sigma, each contour
```

8.74 run

Summary of *Sherpa*/S-Lang module `run` functions.

Most commands issued in *Sherpa* cause it to do tasks that are accomplished quickly. However, several cause *Sherpa* to begin potentially time-consuming tasks, like `FIT`, `PROJECTION`, *etc.* This time consumption could make life difficult for the novice user if he or she was to do something like this:

```
sherpa> projection
[...done 20 minutes later...]
["Hmm, I'd like to save those results in S-Lang scope..."]
sherpa> p = do_projection()   # (does not actually exist)
[...done 20 minutes later...and the user is somewhat disturbed...]
```

To avoid this, the *Sherpa*/S-Lang module utilizes two different kinds of functions: `run` functions that actually perform time-consuming tasks and return the results, and `get` analogues to the `run` functions which simply return the most recent relevant results. Applied to the situation above, the second 20-minute run would be avoided: the results would be carried into S-Lang scope instantly. Or the user could have just typed `run_projection` in the beginning to achieve the same result.

Name	**Description**
`run_fit`	Fits datasets, and retrieves information about the fits
`run_unc`	Determines confidence intervals, retrieves the parameter bounds
`run_proj`	Determines confidence intervals, retrieves the parameter bounds
`run_cov`	Determines confidence intervals, retrieves the parameter bounds
`run_intunc`	Displays statistics as a function of parameter value, and retrieves the value and statistic arrays
`run_intproj`	Displays statistics as a function of parameter value, and retrieves the value and statistic arrays
`run_regunc`	Displays contours of statistics as a function of parameter values, and retrieves arrays of values and statistics
`run_regproj`	Displays contours of statistics as a function of parameter values, and retrieves arrays of values and statistics

Table 8.15: Summary of *Sherpa*/S-Lang Module `run` Functions

8.75 run_fit

Module function to fit datasets, and retrieve information about the final fit.

```
Struct_Type run_fit([{Integer_Type | Array_Type}])
```

Arguments:

(1) Dataset number or array of dataset numbers (default all datasets)

This function initiates a *Sherpa* fit. When complete, it calls `get_goodness`, and a structure or an array of structures is returned. A single structure is returned if only one data set is fit, and an array of N+1 structures is returned if N data sets are fit simultaneously. For the latter case, the summed statistics for the combined fit are returned in the 0th element of the structure array.

See the related *Sherpa* commands `FIT` and `GOODNESS` for more information.

Examples:

1. Load two datasets; fit the second and examine the results:

```
sherpa> DATA 1 example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example2.rmf
ARF is being input from:
  <directory_path>/example2.arf
Background data are being input from:
  <directory_path>/example2_bkg.pha
sherpa> DATA 2 spec.dat
sherpa> () = set_source_expr(2,"CONST[cs]")
sherpa> result = run_fit(2)
 LVMQT: V2.0
 LVMQT: initial statistic value = 82.2297
 LVMQT: final statistic value = 63.9111 at iteration 3
            cs.c0  57.7977

sherpa> print(result)
dataset         =  2
datatype        =  source
stat            =  63.9111
numbins         =  100
dof             =  99
rstat           =  0.645567
qval            =  0.997605
```

8.76 run_paramest

Module functions to run parameter estimation routines and retrieve information.

```
Array_Type run_unc([Array_Type])
Array_Type run_proj([Array_Type])
Array_Type run_cov([Array_Type])
Struct_Type run_intunc(String_Type)
Struct_Type run_intproj(String_Type)
Struct_Type run_regunc(Array_Type)
Struct_Type run_regproj(Array_Type)
```

See `run_unc`, `run_proj`, `run_cov`, `run_intunc`, `run_intproj`, `run_regunc`, and `run_regproj` for more information.

8.76.1 CHANGES IN CIAO 3.2

The `run_intunc()`, `run_intptoj()`, `run_regunc()`, and `run_regproj()` routines have been updated so that they only return one variable. Please see the documentation for the individual functions for further details.

8.76.2 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields. Please see the documentation for the corresponding "get" function for further details.

8.77 run_paramestint

Module functions to display statistics as a function of parameter value, and to retrieve the value and statistic arrays

```
Struct_Type run_intunc(String_Type)
Struct_Type run_intproj(String_Type)
```

Argument:

(1) A string representing an individual thawed model parameter

These functions initiate the `INTERVAL-UNCERTAINTY` and `INTERVAL-PROJECTION` parameter estimation methods respectively. The chosen method is run using the most recently fit datasets, which are automatically determined and hence are not function arguments. When done, each returns a structure, which are the same as those returned by `get_intunc` and `get_intproj`.

These functions can be used to retrieve information similar to that provided by the *XSPEC* command `steppar`.

Examples:

1. Fit a dataset; get information about χ^2 as a function of power-law amplitude `p.ampl`

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> () = set_subtract
sherpa> () = set_source_expr(1,"POW[p]")
sherpa> set_verbose(0)
sherpa> () = run_fit
sherpa> list_intproj()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast             1               1    Switch to LM/simplex: 0(n)/1(y)
expfac           3               3          Expansion factor for grid
```

```
arange           1              1              Auto-range: 0(n)/1(y)
min              0              0                      Minimum value
max              0              0                      Maximum value
log              0              0             Log-spacing: 0(n)/1(y)
nloop           20             20              Number of grid points
sigma            1              1                    Number of sigma
sherpa> sherpa.intproj.sigma = 3
sherpa> sherpa.intproj.nloop = 100
sherpa> intproj = run_intproj("p.ampl")
[...plot displayed...]
sherpa> print(intproj)
x0              =  Float_Type[100]
y               =  Float_Type[100]
name            =  p.ampl
bfit            =  0.000191983
config          =  sherpa_VisParEst_State
sherpa> printarr(intproj.x0,3)
9.19654e-05
9.39859e-05
9.60065e-05
sherpa> printarr(intproj.y,3)
270.566
267.213
263.936
```

The second-to-last call displays the first three values of the `p.ampl` grid, while the last call displays the best-fit statistic given those `p.ampl` values.

8.77.1 CHANGES IN CIAO 3.2

The `run_intunc()` and `run_intproj()` commands no longer fail with an error message when called. This means that you can use

```
retval = run_intproj(parameter_name);
```

rather than having to use `sherpa_eval()` to call `INTERVAL-PROJECTION` (or `INTERVAL-UNCERTAINTY`) and then `get_regproj()` to access the results.

8.77.2 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields: `name`, `bfit`, and `config`. These fields contain information on the name of the parameter, its best-fit value, and the values used by the "interval" command to calculate the `x0` and `y` values.

8.78 run_paramestlim

Module functions to determine confidence intervals, and retrieve the parameter bounds.

```
Array_Type run_unc([Array_Type])
Array_Type run_proj([Array_Type])
Array_Type run_cov([Array_Type])
```

Argument:

(1) An array of strings representing individual thawed model parameters (default all thawed parameters)

These functions initiate runs of the `UNCERTAINTY`, `PROJECTION`, and `COVARIANCE` parameter estimation methods respectively. The chosen method is run using the most recently fit datasets, which are automatically determined and hence are not function arguments. When done, each returns an array of structures, with one structure for each parameter whose bounds were estimated. These structures are the same as those returned by `get_unc`, `get_proj`, and `get_cov`.

Examples:

1. Fit a dataset; run projection and examine the results:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> () = set_subtract
sherpa> () = set_source_expr(1,"POW[p]")
sherpa> set_verbose(0)
sherpa> () = run_fit
sherpa> proj = run_proj
sherpa> print(length(proj))
2
sherpa> print(proj[0])
name            =  p.gamma
val             =  1.79469
vlo             =  1.71462
vhi             =  1.87662
sigma           =  1
sherpa> print(proj[1])
name            =  p.ampl
val             =  0.000191983
vlo             =  0.000180884
vhi             =  0.000203087
sigma           =  1
```

2. Repeat the last example, but only run covariance for one parameter:

```
sherpa> cov = run_cov(["p.ampl"])
sherpa> print(cov[0])
name            =  p.ampl
```

```
val            =  0.000191983
vlo            =  0.000180873
vhi            =  0.000203094
sigma          =  1
```

8.78.1 CHANGES IN CIAO 3.1

The structures returned by these functions contain an additional field, `sigma`, which lists the value of the sigma parameter used in the covariance, uncertainty, or projection run.

8.79 run_paramestreg

Module functions to display contours of statistics as a function of parameter values, and to retrieve arrays of values and statistics

```
Struct_Type run_regunc(Array_Type)
Struct_Type run_regproj(Array_Type)
```

Argument:

(1) An array of two strings representing two thawed model parameters.

These functions initiate the `REGION-UNCERTAINTY` and `REGION-PROJECTION` parameter estimation methods respectively. The chosen method is run using the most recently fit datasets, which are automatically determined and hence are not function arguments. When done, each returns a structure, which are the same as those returned by `get_regunc` and `get_regproj`.

Examples:

1. Fit a dataset; get information about χ^2 as a function of power-law amplitude `p.ampl` and slope `p.gamma`

```
sherpa> () = load_ascii(1,"spec.dat")
sherpa> () = create_model("POLY","p")
sherpa> set_thawed(["p.c1","p.c2","p.c3"])
sherpa> () = set_source_expr(1,"p")
sherpa> () = run_fit
 LVMQT: V2.0
 LVMQT: initial statistic value = 82.2297
 LVMQT: final statistic value = 62.2247 at iteration 3
             p.c0  61.4774
             p.c1  -0.380228
             p.c2  0.00993229
             p.c3  -7.01741e-05
```

```
sherpa> list_regproj()
Parameter   Current         Default                        Description
----------------------------------------------------------------------
fast              1               1    Switch to LM/simplex: 0(n)/1(y)
expfac            3               3          Expansion factor for grid
arange            1               1              Auto-range: 0(n)/1(y)
min           [0,0]           [0,0]          Minimum values, each axis
max           [0,0]           [0,0]          Maximum values, each axis
log           [0,0]           [0,0]  Log-spacing: 0(n)/1(y), each axis
nloop       [10,10]         [10,10]   Number of grid points, each axis
sigma       [1,2,3]         [1,2,3]      Number of sigma, each contour
sherpa> sherpa.regproj.sigma = [1.6,2.6]
sherpa> regproj = run_regproj(["p.c0","p.c1"])
[...plot displayed...]
sherpa> print(regproj)
x0              =  Float_Type[100]
x1              =  Float_Type[100]
y               =  Float_Type[100]
levels          =  Float_Type[2]
name            =  String_Type[2]
bfit            =  Double_Type[2]
config          =  sherpa_VisParEst_State
sherpa> printarr(regproj.x0,3)
61.3661
61.3661
61.3661
sherpa> printarr(regproj.x1,3)
-1.79518
-1.48075
-1.16631
sherpa> printarr(regproj.y,3)
154.651
118.31
90.9931
```

The third-to-last and second-to-last function calls cause the first three values of the `p.c0` and `p.c1` grid axes to be displayed (with `p.c0` repeated because a 2-D grid is being computed). The last function call displays the best-fit statistic given the `p.c0` and `p.c1` values.

8.79.1 CHANGES IN CIAO 3.2

The `run_regunc()` and `run_regproj()` commands no longer place two arrays on the stack. This means that you can use

```
retval = run_regproj(parameter_names);
```

rather than having to say something like

```
(,,retval) = run_regproj(parameter_names);
```

8.79.2 CHANGES IN CIAO 3.1

The structures returned by these functions contain additional fields: `levels`, `name`, `bfit`, and `config`. These fields contain information on the statistic value for each contour, the name of the parameters, their best-fit values, and the values used by the "region" command to calculate the `x0`, `x1`, and `y` values.

8.80 set

Summary of *Sherpa*/S-Lang module functions that change settings or data.

The `set` functions of the *Sherpa*/S-Lang module:

- Move numerical data *into Sherpa* after, *e.g.*, they were manipulated. (These data may be retrieved from *Sherpa* using analogous `get` functions.)
- Change *Sherpa* settings. (These settings may be determined using analogous `get` functions.)

(See below to find out how plot configuration once done using *Sherpa*'s no-longer-existent `SET` is now done with the *Sherpa* state object.)

Name	**Description**
`set_erroff` | `set_xerroff` | `set_yerroff`	Turn off the drawing of errorbars in *Sherpa* plots
`set_erron` | `set_xerron` | `set_yerron`	Turn on the drawing of errorbars in *Sherpa* plots
`set_lin` | `set_xlin` | `set_ylin`	Switch plotting axis to linear scale in *Sherpa* plots
`set_log` | `set_xlog` | `set_ylog`	Switch plotting axis to logarithmic scale in *Sherpa* plots
`set_data` | `set_back`	Assign source and background data
`set_errors` | `set_berrors`	Assign source and background error estimates
`set_syserrors` | `set_bsyserrors`	Assign source and background systematic errors
`set_weights` | `set_bweights`	Assign source and background statistical weights
`set_filter` | `set_bfilter`	Assign source and background filters from an array
`set_groups` | `set_bgroups`	Assign grouping to source and background from an array
`set_quality` | `set_bquality`	Assign quality to source and background from an array
`set_axes` | `set_baxes`	Create new source and background dataspaces
`set_analysis`	Set units for 1-D spectral analysis
`set_coord`	Set the coordinate system for 2-D image analysis
`set_par`	Set model parameter values, *etc.*
`set_thawed` | `set_frozen`	Thaw or freeze model parameter values
`set_ignore` | `set_signore` | `set_bignore`	Set source and background ignore ranges in 1-D datasets

`set_ignore2d` | `set_signore2d` | `set_bignore2d`	Set source and background exclude regions in 2-D datasets
`set_ignore_all` | `set_signore_all` | `set_bignore_all`	Ignore all source and background data
`set_ignore_bad` | `set_signore_bad` | `set_bignore_bad`	Ignore all bad quality source and background data
`set_notice` | `set_snotice` | `set_bnotice`	Set source and background notice ranges in 1-D datasets
`set_notice2d` | `set_snotice2d` | `set_bnotice2d`	Set source and background include regions in 2-D datasets
`set_notice_all` | `set_snotice_all` | `set_bnotice_all`	Notice all source and background data
`set_subtract` | `set_unsubtract`	Do or undo background subtraction
`set_exptime` | `set_bexptime`	Set source and background exposure times
`set_backscale` | `set_bbackscale`	Set source and background extraction region areas
`set_source_expr` | `set_bg_expr`	Set source and background model stack expression
`set_inst_expr` | `set_sinst_expr` | `set_binst_expr`	Set source and background instrument model stack expression
`set_dir`	Set into directory
`set_verbose`	Reset *Sherpa*'s verbosity

Table 8.16: Summary of *Sherpa*/S-Lang Module `set` Functions

The following table shows how settings once made using the `SET` are now made with the *Sherpa* state object:

Old Command:	**New Functions/Settings:**
`SET PLOT NOERRORBARS`	`set_erroff`, or change `x_errorbars` and `y_errorbars` to 0 in the plotting state objects
`SET PLOT XERRORBARS`	`set_xerron` and `set_yerroff`, or change `x_errorbars` to 1 and `y_errorbars` to 0 in the plotting state objects
`SET PLOT YERRORBARS`	`set_xerroff` and `set_yerron`, or change `x_errorbars` to 0 and `y_errorbars` to 1 in the plotting state objects
`SET PLOT ERRORBARS`	`set_erron`, or change `x_errorbars` and `y_errorbars` to 1 in the plotting state objects
`SET PLOT LIN LIN`	`set_lin`, or change `x_log` and `y_log` to 0 in the plotting state objects
`SET PLOT LIN LOG`	`set_xlin` and `set_ylog`, or change `x_log` to 0 and `y_log` to 1 in the plotting state objects
`SET PLOT LOG LIN`	`set_xlog` and `set_ylin`, or change `x_log` to 1 and `y_log` to 0 in the plotting state objects
`SET PLOT LOG LOG`	`set_log`, or change `x_log` and `y_log` to 1 in the plotting state objects
`SET PLOT AUTORANGE`	Leave `sherpa.multiplot.xfilter_lo` and `xfilter_hi` set to `NULL`
`SET PLOT RANGE MIN:MAX`	Set `sherpa.multiplot.xfilter_lo` to `MIN` and `sherpa.multiplot.xfilter_hi` to `MAX`

Table 8.17: Configuring Plots in CIAO 3.x

See the documentation on the *Sherpa* state objects for more information.

8.81 set_analysis

Module function to set the units for 1-D spectral analysis.

```
Integer_Type set_analysis([Integer_Type],String_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) Units string

The `set_analysis()` function tells *Sherpa* the units in which to do subsequent analyses, for the specified dataset. For *Sherpa* version 3.0.2, supported (case-insensitive) string inputs are "bin" (equivalent to `ANALYSIS CHANNELS`), "kev" (equivalent to `ANALYSIS ENERGY`), "ang" (equivalent to `ANALYSIS WAVE`), "ev", "mev", "nm", and "hz".

Examples:

1. ```
   sherpa> data example.pha
   sherpa> instrument = rsp[a](example.rmf,example.arf)
   sherpa> () = set_analysis(1,"hz")
   sherpa> get_analysis()
   Hz
   sherpa>
   ```

8.82 set_axes

Module functions for creating new source or background dataspaces.

```
Integer_Type set_axes([Integer_Type,]Struct_Type|Array_Type)
Integer_Type set_baxes([Integer_Type,]Struct_Type|Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)
```

(2) A structure containing fields that define a 1-D dataspace, or

(2) An array that defines a 1-D dataspace, or

(2) An array of N structures containing fields that define an N-D dataspace, or

(2) An array of N arrays that define an N-D dataspace.

Note that if only one argument is provided, it is assumed to be one of the quantities listed for argument 2, and the dataset is assumed to be dataset 1.

The `set_axes` function is the equivalent of *Sherpa*'s `DATASPACE` command. (The function `set_baxes`, used to set up dataspaces for background datasets, has no *Sherpa* command analogue.) A "dataspace" is an N-dimensional grid representing the independent variables of a dataset (*e.g.*, the x values in $y = f(x)$).

A given axis may be defined using an array, or a structure; the former are appropriate for unbinned data entry, while structures are appropriate for either unbinned or binned data entry (the equivalent of `DATASPACE HISTOGRAM`). The structures must contain fields named `lo` and `hi` (for binned dataspaces), or `mid` (for unbinned dataspaces). These fields point to numeric arrays. (`set_axes` can thus process the structures output by `get_axes`. Note that the array lengths of the `lo` and `hi` fields of each input structure must be equal (though they may vary from axis to axis; *e.g.*, to define a 512×1024 binned dataspace).

The set_axes functions should be used with care, as it instructs Sherpa to replace any dataset previously input with the same dataset number, with the new dataspace.

See the related *Sherpa* command `DATASPACE` for more information.

Examples:

1. Create a new 1-D dataspace:

```
sherpa> DATA spec.dat
sherpa> str = get_axes(1)
sherpa> print(str)
axistype        =  Channels
axisunits       =  unknown
lo              =  NULL
hi              =  NULL
mid             =  Float_Type[100]
sherpa> theaxis = @(foo[0].mid)
sherpa> () = set_axes(1,theaxis)
```

In this example, the user takes a dataset and ends up replacing the read-in counts with zeroes. In the first step, unbinned ASCII data are read in. Because they are unbinned, `get_axes` returns a structure for which only the `mid` field is non-NULL. (There are no lower and upper bin boundaries.) The array pointed to by `str.mid` is then *copied* (the '@' sign denotes a total copy of array contents), and used to define a new dataspace via `set_axes`.

2. Create a new binned 2-D dataspace:

```
sherpa> variable a0 = struct{lo,hi}
```

```
sherpa> a0.lo = [1:1024]
sherpa> a0.hi = [2:1025]
sherpa> variable a1 = struct{lo,hi}
sherpa> a1.lo = [1:1024]
sherpa> a1.hi = [2:1025]
sherpa> a = Struct_Type[2]
sherpa> a[0] = a0
sherpa> a[1] = a1
sherpa> set_axes(1,a)
1
```

In this example, the user creates two structures which are minimally sufficient for setting up a binned dataspace (such a structure must have fields `lo` and `hi`, and it could have more fields, but these are ignored). Arrays are defined for each of the structure fields, such that the first bin has lower and upper bin boundaries 1 and 2, the second bin 2 and 3, *etc.* An array of structures `a` is then defined, and sent to *Sherpa* via `set_axes`. The call returns 1, indicating that *Sherpa* successfully set up a 1024×1024 dataspace.

8.83 set_backscale

Module functions for setting source and background extraction region areas.

```
Integer_Type set_backscale([Integer_Type],Float_Type)
Integer_Type set_bbackscale([Integer_Type],Float_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) Extraction region area

These functions take their names from the PHA header keyword `BACKSCAL`. `BACKSCAL` can be defined as the ratio of the area of the source (or background) extraction region in image pixels to the total number of image pixels. The fact that there is no ironclad definition for this quantity does not matter so long as the `BACKSCAL` for a source dataset and its associated background dataset are defined in the similar manner, because only the *ratio* of source and background `BACKSCAL` is used in analyses.

The specified backscale replaces the current value of the backscale within *Sherpa*, which as mentioned above may have been set when the dataset was loaded from the value of the `BACKSCAL` header keyword of a PHA file.

See the related *Sherpa* commands `SETDATA` and `SETBACK` for more information.

Examples:

1. Change the backscale value of dataset 1 from 0.0441895 to 0.07:

```
sherpa> get_backscale()
0.0441895
sherpa> set_backscale(1,0.07)
1
sherpa> get_backscale()
0.07
```

8.84 set_coord

Module function to set the coordinate system for 2-D image analysis.

```
Integer_Type set_coord([Integer_Type],String_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) Coordinate system string

The `set_coord()` function indicates to *Sherpa* the coordinate system within which to do 2-D image analysis. For *Sherpa* version 3.0.2, supported string inputs are "logical" (equivalent to "image"), "physical", and "world" (equivalent to "wcs").

Examples:

1.
```
sherpa> data example.fits
sherpa> () = set_coord(1,"physical")
sherpa> get_coord()
physical
sherpa>
```

8.85 set_data

Set source and background data using the S-lang module function in Sherpa.

```
Integer_Type set_data([Integer_Type,]Array_Type)
Integer_Type set_back([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) An array of source/background data amplitudes

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The `set_data` and `set_back` functions allow the user to assign new source and background data amplitudes (for example number of counts) (*i.e.*, the y values in $y = f(x)$) to *Sherpa* datasets.

Note that:

- The input array length must match the number of bins in the `filtered` dataset; consequently, if one's goal is to use these functions in data manipulation, it is *important not to change the filter in Sherpa* between any call to, *e.g.*, `get_data` and `set_data`!
- The input array is typecast to match the type of its associated dataspace (see, *e.g.*, set_axes for a definition of dataspaces). For instance, if dataset 3 is of `Double_Type`, then when `set_data(3,<array>)` is run, the array is typecast to `Double_Type` if necessary.

Examples:

1. Retrieve data from *Sherpa*, process it, and replace:

```
sherpa> DATA spec.dat
sherpa> d = get_data(1)
sherpa> print(d)
2
8
12
20
18
27
....
sherpa> d -= 10
sherpa> print(d)
-8
-2
2
10
8
17
....
sherpa> () = set_data(,d)
```

   In this example, data are read into *Sherpa* in the first command and the counts values are retrived using `get_data`. Then, 10 counts are subtracted off every element of the array d; the new array is then sent to *Sherpa*, where it overwrites the old array. print command shows the first numbers of the array values.

8.86 set_dir

Module function to change directories.

```
Integer_Type set_dir(String_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) The directory to set into.

See the related *Sherpa* command `CD` for more information.

8.87 set_erroff

Switch off the drawing of error bars in Sherpa plots

```
set_erroff
set_xerroff
set_yerroff
```

These commands allow a user to turn off the drawing of error bars in all *Sherpa* plots. A plotting command LPLOT needs to be issued in order for these commands to take effect.

Errors bars can be turned on or off in plots (from `LPLOT`) by setting the `x_errorbars` and `y_errorbars` fields in the corresponding Sherpa configuration variable (also called "state object") `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, or `sherpa.resplot`.

The functions `set_erroff`, `set_xerroff`, and `set_yerroff` allow you to change these fields in all the configuration variables (`sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, `sherpa.resplot`) in one go as shown in the following table:

Function Name	Description
`set_erroff`	Set `x_errorbars = 0` and `y_errorbars = 0`
`set_xerroff`	Set `x_errorbars = 0`
`set_yerroff`	Set `y_errorbars = 0`

Examples:

1. `sherpa> set_erroff`

   This turns off the drawing of errorbars (both in x and y directions) in all *Sherpa* plots.

2. `sherpa> set_xerroff`

   This turns off the drawing of errorbars in only the x direction in all *Sherpa* plots.

3. ```
   sherpa> set_yerroff
   sherpa> print(sherpa.fitplot.y_errorbars)
   0
   sherpa> lplot 2 fit delchi
   ```

   The first command (`set_yerroff`) turns off the drawing of errorbars in only the y direction in all Sherpa plots. The second command prints out the value of the `y_errorbars` field of the `sherpa.fitplot` variable; it is set to 0 because of the `set_yerroff` command. The third command plots fit and delchi without y errorbars.

4. ```
   sherpa> set_yerroff
   sherpa> sherpa.fitplot.y_errorbars = 1
   sherpa> lplot fit
   ```

   The first command (`set_yerroff`) turns off the drawing of errorbars in only the y direction in all *Sherpa* plots. The second command changes the `sherpa.fitplot` variable so that its y_errorbars field is set to 1 which means that "`LPLOT FIT`" will include y error bars. The third command plots fit with y errorbars.

8.88 set_erron

Switch on the drawing of error bars in Sherpa plots

```
set_erron
set_xerron
set_yerron
```

These commands allow a user to turn on the drawing of error bars in all *Sherpa* plots. A plotting command LPLOT needs to be issued in order for these commands to take effect.

Errors bars can be turned on or off in plots (from `LPLOT`) by setting the `x_errorbars` and `y_errorbars` fields in the corresponding Sherpa configuration variable (also called "state object") `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, or `sherpa.resplot`.

The functions `set_erron`, `set_xerron`, and `set_yerron` allow you to change these fields in all the configuration variables (`sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, `sherpa.resplot`) in one go as shown in the following table:

Function Name	Description
set_erron()	Set x_errorbars = 1 and y_errorbars = 1
set_xerron()	Set x_errorbars = 1
set_yerron()	Set y_errorbars = 1

Examples:

1. `sherpa> set_erron`

   This turns on the drawing of errorbars (both in x and y directions) in all *Sherpa* plots.

2. `sherpa> set_xerron`

   This turns on the drawing of errorbars in only the x direction in all *Sherpa* plots.

3. ```
   sherpa> set_yerron
   sherpa> print(sherpa.fitplot.y_errorbars)
   1
   sherpa> lplot 2 fit delchi
   ```

   The first command (**set_yerron**) turns on the drawing of errorbars in only the y direction in all *Sherpa* plots. The second command prints out the value of the **y_errorbars** field of the **sherpa.fitplot** variable; it is set to 1 because of the **set_yerron** command. The third command displays two plots with the errorbars in only the y direction.

4. ```
   sherpa> set_yerron
   sherpa> sherpa.fitplot.y_errorbars = 0
   sherpa> lplot fit
   ```

   The first command (**set_yerron**) turns on the drawing of errorbars in only the y direction in all *Sherpa* plots. The second command changes the **sherpa.fitplot** variable so that its y_errorbars field is set to 0 which means that "`LPLOT FIT`" will not include y error bars. The third command displays the fit with no errorbars in y direction.

8.89 set_errors

Module functions for assigning source and background error estimates.

```
Integer_Type set_errors([Integer_Type,]Array_Type)
Integer_Type set_berrors([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) An array of source/background error estimates

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The **set_errors** and **set_berrors** functions allow the user to assign new source and background error estimates to *Sherpa* datasets.

Note that:

- Any errors that are assigned via `set_errors` replace the error estimates that *Sherpa* makes automatically using the current `STATISTIC` setting, and are used until the errors are explicitly redefined.
- The input array length must match the number of bins in the `filtered` dataset; consequently, if one's goal is to use these functions in data manipulation, it is *important not to change the filter in Sherpa* between any call to, *e.g.*, `get_errors` and `set_errors`!
- The input array is typecast to match the type of its associated dataspace (see, *e.g.*, set_axes for a definition of dataspaces). For instance, if dataset 3 is of `Double_Type`, then when `set_errors(3,<array>)` is run, the array is typecast to `Double_Type` if necessary.

See the related *Sherpa* commands `ERRORS` and `BERRORS` for more information.

Examples:

1. Cause *Sherpa* to perform a least-squares fit:

```
sherpa> DATA spec.dat
sherpa> d = get_data()
sherpa> d[*] = 1
sherpa> () = set_errors(,d)
```

In this example, an array of the length of the filtered data is retrieved from *Sherpa*, and all elements of the array are reset to 1. This new array is then sent to *Sherpa*, where it overwrites the old array of error estimates.

8.90 set_exptime

Module functions for setting source and background exposure times.

```
Integer_Type set_exptime([Integer_Type],Float_Type)
Integer_Type set_bexptime([Integer_Type],Float_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) Exposure time

The specified exposure time replaces the current value of the exposure time within *Sherpa*, which may have been set when the dataset was loaded from the value of the `EXPTIME` header keyword of a PHA file.

See the related *Sherpa* commands `SETDATA` and `SETBACK` for more information.

Examples:

1. Set the exposure time of the background dataset of dataset 4 to 30 ks:

```
sherpa> get_bexptime(4)
108676
sherpa> () = set_bexptime(4,30000.0)
sherpa> get_bexptime(4)
30000
```

8.91 set_filter

Module functions for assigning source and background filters from an array.

```
Integer_Type set_filter([Integer_Type,]Array_Type)
Integer_Type set_bfilter([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) An array of source/background filter values

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The `set_filter` and `set_bfilter` functions allow the user to set new filters for source and background datasets.

The input array is a `Short_Type`, `Long_Type`, or `Integer_Type` array of the same length as the input (*unfiltered*) dataset for which a new filter is to be defined. If an input array element is zero, then the corresponding data bin is to be ignored; non-zero, and the corresponding bin is to be noticed/used.

One may use these functions in combination with `get_filter` and `get_bfilter` to define new methods of data filtering that go beyond those currently coded in *Sherpa*; *e.g.*, filtering based on data amplitude. See example below.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a 100-bin dataset, and filter out bins 10 to 60:

```
sherpa> DATA spec.dat
sherpa> a = [0:99]
sherpa> a[*] = 1
```

```
sherpa> a[[9:59]] = 0
sherpa> () = set_filter(1,a)
```

In this example, data are read into *Sherpa*, and then the filter array `a` defined: step 1 defines a 100-bin-length array, step 2 resets all elements to 1, and step 3 resets a subset of the elements to 0 (bins to be ignored). Note that while the bins are numbered 1 to 100 in the data, the analogous array indices are 0 to 99, so array elements 9 to 59 are reset. The array is then sent to *Sherpa*, where it sets the new filter.

2. Read in a 100-bin dataset, and filter out all bins with counts greater than 65:

```
sherpa> DATA spec.dat
sherpa> WRITE DATA
Write X-Axis: Bin  Y-Axis: Flux (Counts)
          1         59
          2         46
          3         49
          4         65
          5         60
          6         60
          7         74
...
sherpa> d = get_data()
sherpa> a = where(d>65)
sherpa> printarr(a)
6
9
11
26
...
sherpa> f = get_filter()
sherpa> f[a] = 0
sherpa> set_filter(1,f)
1
sherpa> WRITE DATA
Write X-Axis: Bin  Y-Axis: Flux (Counts)
          1         59
          2         46
          3         49
          4         65
          5         60
          6         60
          8         58
...
```

In this example, data are read into *Sherpa*, and the counts amplitude is retrieved using `get_data`. A new array `a` is defined whose elements are the bins for which the data are greater than 65; the length of this array is 13 (13 bins have data greater than 65 and are thus to be ignored). The filter array is then retrieved with `get_filter` and the appropriate 13 elements of that array are reset to 0. The new filter array is then read back into *Sherpa*, and we see, *e.g.*, that bin 7 is now ignored.

8.92 set_groups

Module functions for grouping and setting quality to source and background files from an array.

```
Integer_Type set_groups([Integer_Type,]Array_Type)
Integer_Type set_bgroups([Integer_Type,]Array_Type)
Integer_Type set_quality([Integer_Type,]Array_Type)
Integer_Type set_bquality([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1).

(2) An array defining the grouping or quality and it should be of the same length as the input data set. This argument is required.

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The `set_(b)groups` and `set_(b)quality` functions allow the user to set new groups or quality for source and background datasets.

The input array is a `Integer_Type` array of the same length as the input (*ungrouped*) dataset for which a new grouping or quality scheme is to be defined. An input grouping array element set to -1 marks the beginning of a group, while an input array element set to 1 marks members of that group, so the corresponding bins are treated as one during fitting.

A quality array contains the quality flags for each group: 0 for good (grouped) data; 5 for data labeled as bad by the user (within a tab), and 2 for data labeled as questionable by dmgroup (incomplete groups, *etc.*).

The grouping and quality definitions are based on OGIP standard.

See the related *Sherpa* commands `GROUP` and `QUALITY` for more information.

Examples:

1. Apply a grouping definition from the other PHA file to the data set.

```
sherpa> DATA spec.pha
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
```

```
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

.......

sherpa> g=readfile("grouped.pha")
sherpa> print(g)
_filename       =  grouped.pha
_path           =  /data
_filter         =  NULL
_filetype       =  7
_header         =  String_Type[361]
_exptime        =  13804
_ncols          =  13
_nrows          =  1024
channels        =  Float_Type[596]
counts          =  Float_Type[596]
grouping        =  Integer_Type[1024]
qualityflags    =  Integer_Type[596]
phachans        =  Integer_Type[596]
areascal        =  1
backscal        =  4.2132e-07
errors          =  Float_Type[596]
background      =  /data/bg.pha
arf             =  /data/spec.arf
response        =  /data/spec.rmf
numgroups       =  596
numchans        =  1024

sherpa> set_groups(1,g.grouping)
sherpa> show
...
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 596 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231
.....
```

In this example, ungrouped and group data are read into *Sherpa*, and then group information is retrieved from the other file, by reading the file first into a variable `g` using `readfile`. A new array `g.grouping` can be apply to the ungrouped data set. `g.grouping` is defined by the grouping in the dataset 2. This grouping scheme is then apply to the ungrouped data set with `set_groups`.

2. Apply a grouping scheme from the other data set to ungrouped data.

```
sherpa> DATA spec.pha
```

```
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

.......
sherpa> DATA 2 spec_grp.pha
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231


......

Data 2: spec_grp.pha pha.
Total Size: 131 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231
......
sherpa> g=get_groups(2)
sherpa> print(g)
1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
1
```

```
-1
.....
sherpa> set_groups(1,g)
WARNING: any applied filters are being deleted!
1
sherpa> show

Data 1: spec.pi pha.
Total Size: 131 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231
.....
```

In this example, ungrouped and group data are read into *Sherpa*, and then group information is retrieved using **get_groups** from the grouped dataset. A new array **g** is defined whose elements are defined by the grouping of the dataset 2. This grouping scheme is then apply to the ungrouped data set with **set_groups**.

3. Apply a grouping scheme from the source data set to ungrouped background data.

```
sherpa> DATA spec.pha
sherpa> show
....
-----------------
Input data files:
-----------------

Data 1: spec.pha pha.
Total Size: 1024 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

 Background 1: bg.pha pha.
  Total Size: 1024 bins (or pixels)
  Dimensions: 1
  Total counts (or values): 662

.......

sherpa> g=get_groups(1)
sherpa> print(g)
1
-1
-1
-1
-1
-1
-1
-1
-1
```

```
-1
-1
-1
-1
-1
-1
-1
1
-1
.....
sherpa> set_bgroups(1,g)
WARNING: any applied filters are being deleted!
1
sherpa> show

Data 1: spec.pi pha.
Total Size: 131 bins (or pixels)
Dimensions: 1
Total counts (or values): 2231

 Background 1: bg.pha pha.
 Total Size: 131 bins (or pixels)
  Dimensions: 1
  Total counts (or values): 662
......
```

In this example, group data and ungrouped background data are read into *Sherpa*, and then group information is retrieved using **get_groups** from the grouped dataset. A new array **g** is defined whose elements are defined by the grouping of the source data set. This grouping scheme is then apply to the ungrouped background data set with **set_bgroups**.

8.93 set_ignore

Module functions for ignoring source and background filter ranges in 1-D datasets.

```
Integer_Type set_ignore([Integer_Type],Float_Type,Float_Type)
Integer_Type set_signore([Integer_Type],Float_Type,Float_Type)
Integer_Type set_bignore([Integer_Type],Float_Type,Float_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) The lower bound for the ignore range (default no lower bound)

(3) The upper bound for the ignore range (default no upper bound)

The commands `set_ignore`, `set_signore`, and `set_bignore` allow the user to specify data ranges to ignore in both source and background datasets, a source dataset, or a background dataset respectively. These functions may only be applied to 1-D data. Use `set_ignore2d` to set `exclude` regions in images.

Note that these functions work in the space given by the current `ANALYSIS` setting, *e.g.*, if the setting is `ENERGY`, then the defined bounds are assumed to be in energy units. Note that in CIAO 3.1 `set_ignore` will work only if instrument models for both source and background are set.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a PHA file with instrument; ignore the lowest-energy channels:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> set_ignore(1,,0.3)
1
sherpa> ANALYSIS
Analysis Space for Dataset 1: Energy
```

   In this example, all data below 0.3 keV are ignored in dataset 1.

2. Continue Example 1, ignoring all data between 6 and 12 Å:

```
sherpa> ANALYSIS WAVE
sherpa> set_ignore(,6,12)
1
```

3. Continue Example 2, now ignoring all source data above 9 keV:

```
sherpa> ANALYSIS ENERGY
sherpa> set_signore(1,9,)
1
```

   The corresponding background data above 9 keV are still noticed.

8.94 set_ignore2d

Module functions for ignoring source and background regions in 2-D datasets.

```
Integer_Type set_ignore2d([Integer_Type],String_Type[,String_Type])
Integer_Type set_signore2d([Integer_Type],String_Type[,String_Type])
Integer_Type set_bignore2d([Integer_Type],String_Type[,String_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) A filter expression: <`filterExpr`>

(3) The coordinate system expression

The commands `set_ignore2d`, `set_signore2d`, and `set_bignore2d` allow the user to specify regions to ignore in both source and background datasets, a source dataset, or a background dataset respectively. These functions may only be applied to 2-D data. Use `set_ignore` to set ranges to ignore in 1-D spectra.

See the *Sherpa Filtering* Chapter for the full definition of <`filterExpr`>. (Also see examples below.)

The default coordinate system expression is "logical" (*i.e.*, image coordinates). Allowed coordinate system expressions include:

- "logical" | "image" - image coordinates
- "physical" - physical coordinates
- "wcs" | "world" - world coordinates

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in an image; ignore the central source using a circular `exclude` region:

```
sherpa> load_image(2,"example_img.fits")
1
sherpa> set_ignore2d(2,"circle(247,257,20)")
1
```

   In this example, all data within a radius of 20 pixels of the image coordinate (247,257) are ignored.

2. Apply a box filter to an image in physical coordinates:

```
sherpa> load_image(1,"example_img2.fits")
1
sherpa> set_ignore2d(,"box(4050,3920,30,30)","physical")
1
```

8.95 set_ignore_all

Module functions for ignoring all source and background data.

```
Integer_Type set_ignore_all([Integer_Type])
Integer_Type set_signore_all([Integer_Type])
Integer_Type set_bignore_all([Integer_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

The commands `set_ignore_all`, `set_signore_all`, and `set_bignore_all` allow the user to ignore all data in both source and background datasets, a source dataset, or a background dataset respectively.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a PHA file with instrument; ignore all channels as a prelude to defining notice ranges:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> set_ignore_all(1)
1
...
```

8.96 set_ignore_bad

Module functions for ignoring bad (quality > 0) channels.

```
Integer_Type set_ignore_bad([Integer_Type])
Integer_Type set_signore_bad([Integer_Type])
Integer_Type set_bignore_bad([Integer_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

The commands `set_ignore_bad`, `set_signore_bad`, and `set_bignore_bad` allow the user to ignore bad quality (quality > 0) channels in both source and background datasets, a source dataset, or a background dataset respectively. These functions have no impact if the data quality has not been defined, either within a column of a PHA dataset or via the `READ QUALITY` command.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a PHA file; ignore the bad channels:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> f = get_filter()
sherpa> print(length(where(f>0)))
95                                        # 95 noticed bins
sherpa> set_ignore_bad
1
sherpa> f = get_filter()
sherpa> print(length(where(f>0)))
80                                        # now only 80 noticed bins
```

8.97 set_lin

Switch plotting axis to linear scale in Sherpa plots

```
set_lin
set_xlin
set_ylin
```

These commands allow a user to switch x- and y-axis to linear scale in all *Sherpa* plots. A plotting command LPLOT needs to be issued in order for these commands to take effect.

A user can control log and linear scale of the x and y-axis created with the LPLOT command by setting the **x_log** and **y_log** in the corresponding *Sherpa* configuration variable (also called "state object") **sherpa.plot**, **sherpa.dataplot**, **sherpa.fitplot**, or **sherpa.resplot**.

The functions **set_lin**, **set_xlin**, and **set_ylin** allow you to change these fields in all the configuration variables (**sherpa.plot**, **sherpa.dataplot**, **sherpa.fitplot**, **sherpa.resplot**) at once as shown in the following table:

Function Name	Description
set_lin	Set x_log = 0 and y_log = 0
set_xlin	Set x_log = 0
set_ylin	Set y_log = 0

Examples:

1. **sherpa> set_lin**

   This sets plot axis (both in x and y) to linear scale in all *Sherpa* plots.

2. **sherpa> set_xlin**

   This sets plot axis to linear scale for only the x-axis in all *Sherpa* plots.

3. **sherpa> set_ylin**

```
sherpa> print(sherpa.fitplot.y_log)
0
sherpa> lplot fit
```

The first command (`set_ylog`) sets y-axis to linear scale in all *Sherpa* plots. The second command prints out the value of the `y_log` field of the `sherpa.fitplot` variable; it is set to 0 (false) because of the `set_ylin` command. The third command displays fit in linear scale

8.98 set_log

Switch axis to log in Sherpa plots

```
set_log
set_xlog
set_ylog
```

These commands allow a user to switch x- and y-axis to log in all *Sherpa* plots. A plotting command LPLOT needs to be issued in order for these commands to take effect.

A user can control log and linear scale of the x and y-axis created with the LPLOT command by setting the `x_log` and `y_log` in the corresponding *Sherpa* configuration variable (also called "state object") `sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, or `sherpa.resplot`.

The functions `set_log`, `set_xlog`, and `set_ylog` allow you to change these fields in all the configuration variables (`sherpa.plot`, `sherpa.dataplot`, `sherpa.fitplot`, `sherpa.resplot`) at once as shown in the following table:

Function Name	Description
set_log	Set x_log = 1 and y_log = 1
set_xlog	Set x_log = 1
set_ylog	Set y_log = 1

Examples:

1. `sherpa> set_log`

   This sets plot axis (both in x and y) to log scale in all *Sherpa* plots.

2. `sherpa> set_xlog`

   This sets plot axis to log scale for only the x-axes in all *Sherpa* plots.

3. ```
   sherpa> set_ylog
   sherpa> print(sherpa.fitplot.y_log)
   1
   sherpa> lplot fit
   ```
   ```

The first command (`set_ylog`) sets y-axis to log scale in all *Sherpa* plots. The second command prints out the value of the `y_log` field of the `sherpa.fitplot` variable; it is set to 1 because of the `set_ylog` command. The third command displays fit in log scale

8.99 set_notice

Module functions for noticing source and background filter ranges in 1-D datasets.

```
Integer_Type set_notice([Integer_Type],Float_Type,Float_Type)
Integer_Type set_snotice([Integer_Type],Float_Type,Float_Type)
Integer_Type set_bnotice([Integer_Type],Float_Type,Float_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) The lower bound for the notice range (default no lower bound)

(3) The upper bound for the notice range (default no upper bound)

The commands `set_notice`, `set_snotice`, and `set_bnotice` allow the user to specify data ranges to notice in both source and background datasets, a source dataset, or a background dataset respectively. These functions may only be applied to 1-D data. Use `set_notice2d` to set `include` regions in images.

Note that these functions work in the space given by the current `ANALYSIS` setting, *e.g.*, if the setting is `ENERGY`, then the defined bounds are assumed to be in energy units. Note that in CIAO 3.1 `set_notice` will work only if instrument models for both source and background are set.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a PHA file with instrument; notice the data below 1 keV:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> set_notice(1,,1)
1
sherpa> ANALYSIS
Analysis Space for Dataset 1: Energy
```

2. Continue Example 1, notice all data between 3 and 6 Å:

```
sherpa> ANALYSIS WAVE
sherpa> set_notice(,3.0,6.0)
1
```

3. Continue Example 2, now noticings all source data above 9 keV:

   ```
   sherpa> ANALYSIS ENERGY
   sherpa> set_snotice(1,9.0,)
   1
   ```

   The corresponding background data above 9 keV are still ignored.

8.100 set_notice2d

Module functions for noticing source and background regions in 2-D datasets.

```
Integer_Type set_notice2d([Integer_Type],String_Type[,String_Type])
Integer_Type set_snotice2d([Integer_Type],String_Type[,String_Type])
Integer_Type set_bnotice2d([Integer_Type],String_Type[,String_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) A filter expression: <`filterExpr`>

(3) The coordinate system expression

The commands `set_notice2d`, `set_snotice2d`, and `set_bnotice2d` allow the user to specify regions to notice in both source and background datasets, a source dataset, or a background dataset respectively. These functions may only be applied to 2-D data. Use `set_notice` to set ranges to notice in 1-D spectra.

See the *Sherpa Filtering* Chapter for the full definition of <`filterExpr`>. (Also see examples below.)

The default coordinate system expression is “logical” (*i.e.*, image coordinates). Allowed coordinate system expressions include:

- ”logical” | “image” - image coordinates
- ”physical” - physical coordinates
- ”wcs” | “world” - world coordinates

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in an image; notice the central source using a circular `include` region:

```
sherpa> load_image(2,"example_img.fits")
1
sherpa> set_notice2d(2,"circle(247,257,20)")
1
```

In this example, all data within a radius of 20 pixels of the image coordinate (247,257) are noticed.

2. Apply a box filter to an image in physical coordinates:

```
sherpa> load_image(1,"example_img2.fits")
1
sherpa> set_notice2d(,"box(4050,3920,30,30)","physical")
1
```

8.101 set_notice_all

Module functions for noticing all source and background data.

```
Integer_Type set_notice_all([Integer_Type])
Integer_Type set_snotice_all([Integer_Type])
Integer_Type set_bnotice_all([Integer_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

The commands `set_notice_all`, `set_snotice_all`, and `set_bnotice_all` allow the user to notice all data in both source and background datasets, a source dataset, or a background dataset respectively.

See the related *Sherpa* commands `NOTICE` and `IGNORE` for more information.

Examples:

1. Read in a PHA file with instrument; ignore all channels, then notice all channels:

```
sherpa> () = load_dataset(1,"example.pha")
sherpa> set_ignore_all(1)
1
sherpa> set_notice_all(1)
1
sherpa> SHOW
...
Current filters for dataset 1:
notice source 1 all
Noticed filter size: 95 bins
...
```

8.102 set_par

Module function for setting model parameter values, *etc.*

```
Integer_Type set_par(Struct_Type)
Integer_Type set_par(String_Type,String_Type,{Float_Type | String_Type})
```

Success/Error Return Values: 1/0

Arguments:

(1) A parameter information structure

or

(1) The parameter name

(2) The field to set

(3) The new value for that field

The first form of this function assumes as input the parameter information structure output by **get_par**, *e.g.*

```
sherpa> GAUSS[g]
sherpa> foo = get_par("g.pos")
sherpa> print(foo)
name             =  g.pos
model            =  gauss1d
type             =  src
value            =  0
min              =  -3.40282e+38
max              =  3.40282e+38
delta            =  -1
units            =  NULL
frozen           =  0
linked           =  0
linkexpr         =  NULL
```

The second form of this function takes the arguments listed above; fields to set include:

Argument	Description
`value`	the parameter value (either a number, or a string filename)
`min`	the current (soft) lower bound on allowed parameter values
`max`	the current (soft) upper bound on allowed parameter values
`delta`	the initial step size for the parameter in fits (or -1 to use the default step size)
`frozen`	if 1, freeze the parameter's value; if 0, thaw it

`thawed`	if 1, thaw the parameter's value; if 0, freeze it

Table 8.22: Allowed **set_par** Arguments

See the *Sherpa* command `CREATE` for more information.

Examples:

1. Get a parameter structure; change two fields; set back into *Sherpa*:

```
sherpa> GAUSS[g]
sherpa> foo = get_par("g.pos")
sherpa> print(foo)
name            =  g.pos
model           =  gauss1d
type            =  src
value           =  0
min             =  -3.40282e+38
max             =  3.40282e+38
delta           =  -1
units           =  NULL
frozen          =  0
linked          =  0
linkexpr        =  NULL
sherpa> foo.value = 15.5
sherpa> foo.min = 0.0
sherpa> () = set_par(foo)
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed         10 1.1755e-38 3.4028e+38
 2    pos thawed       15.5          0 3.4028e+38
 3   ampl thawed          1 -3.403e+38 3.4028e+38
```

2. Change two parameter attributes directly:

```
sherpa> GAUSS[g]
sherpa> () = set_par("g.pos","value",15.5)
sherpa> () = set_par("g.pos","min",0.0)
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm thawed         10 1.1755e-38 3.4028e+38
 2    pos thawed       15.5          0 3.4028e+38
 3   ampl thawed          1 -3.403e+38 3.4028e+38
```

8.103 set_paramset

Module functions to thaw or freeze the specified parameter(s).

```
Integer_Type set_thawed([String_Type])
Integer_Type set_frozen([String_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Model name, or model parameter name (default all model parameters)

If:

- no argument is provided, these functions will thaw or freeze *all* model parameters;
- a model component name is provided, these functions will thaw or freeze all parameters of that model; and
- if a model parameter name is provided, these functions will thaw or freeze that parameter.

See the related *Sherpa* commands `THAW` and `FREEZE` for more information.

Examples:

1. Freeze all parameters of a specified Gaussian model, then thaw the position parameter:

```
sherpa> PARAMPROMPT OFF
sherpa> GAUSS[g]
sherpa> set_frozen("g")
1
sherpa> set_thawed("g.pos")
1
sherpa> SHOW g
gauss1d[g]  (integrate: on)
    Param   Type      Value        Min        Max                 Units
    -----   ----      -----        ---        ---                 -----
 1   fwhm frozen         10 1.1755e-38 3.4028e+38
 2    pos thawed          0 -3.403e+38 3.4028e+38
 3   ampl frozen          1 -3.403e+38 3.4028e+38
```

8.104 set_stackexpr

Module functions to set a model stack expression.

```
Integer_Type set_source_expr([Integer_Type,]String_Type])
Integer_Type set_bg_expr([Integer_Type,]String_Type])
Integer_Type set_inst_expr([Integer_Type,]String_Type])
Integer_Type set_sinst_expr([Integer_Type,]String_Type])
Integer_Type set_binst_expr([Integer_Type,]String_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) A string expression

Note that if only one argument is provided, it is assumed to be a string expression, and the dataset is assumed to be dataset 1.

These functions set model stack expressions; `set_sinst_expr` and `set_binst_expr` set instrument model stack expressions for the source dataset only, and the background dataset only, respectively.

See the related *Sherpa* commands `SOURCE`, `BACKGROUND`, and `INSTRUMENT` for more information.

NOTE: In CIAO 3.1 the instrument model stack expression for the background dataset is required for simultanous filtering in energy/wave or fitting of PHA files.

Examples:

1. Specify the source model stack expression for dataset 3:

```
sherpa> PARAMPROMPT OFF
sherpa> GAUSS[g]
sherpa> POW[p]
sherpa> () = set_source_expr(3,"p+g")
```

2. Specify a source model stack expression for dataset 1:

```
sherpa> () = set_source_expr("xswabs[xs1]*(pow[p]+gauss[g])")
sherpa> get_source_expr()
(xs1 * (p + g))
```

8.105 set_subtract

Module functions to subtract background from a source dataset, or to undo subtraction.

```
Integer_Type set_subtract([Integer_Type])
Integer_Type set_unsubtract([Integer_Type])
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

These functions return 1 if successful, and 0 if the (un)subtraction cannot be done (because, *e.g.*, background already have been subtracted before a call to `set_subtract`).

See the related *Sherpa* commands `SUBTRACT` and `UNSUBTRACT` for more information.

Examples:

1. Subtract the background from dataset 1; then add it back in:

```
sherpa> DATA example.pha
The inferred file type is PHA.  If this is not what you want, please
specify the type explicitly in the data command.
WARNING: using systematic errors specified in the PHA file.
RMF is being input from:
  <directory_path>/example2.rmf
ARF is being input from:
  <directory_path>/example2.arf
Background data are being input from:
  <directory_path>/example2_bkg.pha
sherpa> is_subtracted
0
sherpa> set_subtract
1
sherpa> is_subtracted(1)
1
sherpa> set_unsubtract()
1
sherpa> is_subtracted(1)
0
```

The 0 returned from the first call indicates that no, dataset 1 is not yet background subtracted. (Unlike *XSPEC*, *Sherpa* does not automatically subtract background data upon input.) The 1 returned from the third call, issued after the call to the `set_subtract` function, indicates that yes, dataset 1 is now background-subtracted. Similarly, the 0 returned from the last call to `is_subtracted` indicates that the call to `set_unsubtract` was successful.

8.106 set_syserrors

Module functions for assigning source and background systematic error estimates.

```
Integer_Type set_syserrors([Integer_Type,]Array_Type)
Integer_Type set_bsyserrors([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) An array of source/background systematic error estimates

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The `set_syserrors` and `set_bsyserrors` functions allow the user to assign new source and background systematic error estimates to *Sherpa* datasets.

Note that:

- Systematic errors are added in quadrature to statistical errors (error estimates that *Sherpa* makes automatically using the current `STATISTIC` setting) to yield overall error estimates in each bin.
- The input array length must match the number of bins in the `filtered` dataset; consequently, if one's goal is to use these functions in data manipulation, it is *important not to change the filter in Sherpa* between any call to, *e.g.*, `get_syserrors` and `set_syserrors`!
- The input array is typecast to match the type of its associated dataspace (see, *e.g.*, set_axes for a definition of dataspaces). For instance, if dataset 3 is of `Double_Type`, then when `set_syserrors(3,<array>)` is run, the array is typecast to `Double_Type` if necessary.

See the related *Sherpa* commands `SYSERRORS` and `BSYSERRORS` for more information.

Examples:

1. Assign systematic errors of one-tenth the data amplitude to each data bin:

```
sherpa> DATA spec.dat
sherpa> d = get_data()
sherpa> d[*] = 0.1
sherpa> () = set_syserrors(,d)
```

   In this example, an array of the length of the filtered data is retrieved from *Sherpa*, and all elements of the array are reset to 0.1 (the systematic error estimate). This new array is then sent to *Sherpa*, where it overwrites the old array of systematic error estimates.

8.107 set_verbose

Module function to reset *Sherpa*'s verbosity.

```
Void_Type set_verbose(Integer_Type)
```

Arguments:

(1) The verbosity level.

This function sets the amount that *Sherpa* outputs to the screen.

The current verbosity settings are:

Setting	**Displays:**
0	(almost) no output (useful for scripts/batch jobs)
1	normal output; many warnings are suppressed and the results of each fit iteration are not shown (the default)
2	full output

Table 8.23: Verbosity Settings for `set_verbose`

8.108 set_weights

Module functions for assigning source and background statistical weights.

```
Integer_Type set_weights([Integer_Type,]Array_Type)
Integer_Type set_bweights([Integer_Type,]Array_Type)
```

Success/Error Return Values: 1/0

Arguments:

(1) Dataset number (default 1)

(2) An array of source/background statistical weight settings

Note that if only one argument is provided, it is assumed to be an array, and the dataset is assumed to be dataset 1.

The `set_weights` and `set_bweights` functions allow the user to assign new source and background statistical weight values to *Sherpa* datasets. The default value of a statistical weight is 1, meaning the all data bins have equal influence in the calculation of a statistic. A weight less than 1 decreases a bin's contribution to the total statistic value; a weight greater than 1 increases the contribution.

Note that:

- the input array length must match the number of bins in the `filtered` dataset; consequently, if one's goal is to use these functions in data manipulation, it is *important not to change the filter in Sherpa* between any call to, *e.g.*, `get_weight` and `set_weight`!
- the input array is typecast to match the type of its associated dataspace (see, *e.g.*, set_axes for a definition of dataspaces). For instance, if dataset 3 is of `Double_Type`, then when `set_weight(3,<array>)` is run, the array is typecast to `Double_Type` if necessary.

See the related *Sherpa* command `READ WEIGHTS` for more information.

Examples:

1. Change the statistical weights of the lowest-value bins in a dataset to reduce the influence of these bins in a statistic calculation:

```
sherpa> DATA spec.dat
sherpa> SOURCE = CONST[cs]
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 82.2297
 LVMQT: final statistic value = 63.9111 at iteration 3
            cs.c0  57.7977

sherpa> d = get_data(1)
sherpa> d[*] = 1
sherpa> d[[0:5]] = 0.5
sherpa> printarr(d,10)
0.5
0.5
0.5
0.5
0.5
0.5
1
1
...
sherpa> set_weights(1,d)
1
sherpa> FIT
 LVMQT: V2.0
 LVMQT: initial statistic value = 61.7966
 LVMQT: final statistic value = 61.7912 at iteration 2
            cs.c0  57.8627
```

In this example, data are read into *Sherpa* and the counts amplitude array is retrived using `get_data`. Every element of this array is reset to 1, then the first six elements are reset to 0.5. The array is then sent to *Sherpa*, where it overwrites the old weight array; the subsequent fit shows a slight change.

REFERENCES

D. Dewey, H.L. Marshall, K.A. Flanagan, C. Baluta, C.R. Canizares, D.S. Davis, J.E. Davis, T.T. Fang, D.P. Huenemoerder, J.H. Kastner, N.S. Schulz, M.W. Wise, J.J. Drake, J.Z. Juda, M. Juda, A.C. Brinkman, T. Gunsing, J. Kaastra, G. Hartner and P. Predehl, "Towards the Callibration of the HETG Effective Area", *these proceedings*, (1997).

K.A. Flanagan, D. Dewey, L. Bordzol, "Calibration and Characterization of HETG grating elements at the MIT X-ray Grating Evaluation Facility", *SPIE Proceedings* **2518**, 438–456 (1995).

K.A. Flanagan, T.T. Fang, C. Baluta, J.E. Davis, D. Dewey, T.H. Markert, D.E. Graessle, J. Drake, J.E. Fitch, J.Z. Juda, J. Woo, S. Kraft, P. Bulicke, R. Fliegauf, F. Scholze, and G. Ulm, "Modeling the Diffraction Efficiencies of the AXAF High Energy Transmission Gratings: II", *EUX, X-Ray, and Gamma-Ray Instrumentation for Astronomy VII*, Proc. SPIE, Vol. **2808**, 650–676 (1996).

B.L. Henke, E.M. Gullikson, and J.C. Davis, "X-ray Interactions: Photoabsorption, Scattering, Transmission, and Reflection at E = 50–30,000 eV, A = 1–92", *Atomic Data and Nuclear Data Tables*, Vol. 54, No. 2, July 1993.

T.H. Markert, D. Dewey, J.E. Davis, K.A. Flanagan, D.E. Graessle, J.M. Bauer, C.S. Nelson, "Modeling the Diffraction Efficiencies of the AXAF High Energy Transmission Gratings", *SPIE Proceedings* **2518**, 424–437 (1995).

H.L. Marshall, D. Dewey, K.A. Flanagan, C. Baluta, C.R. Canizares, D.S. Davis, J.E. Davis, T.T. Fang, D.P. Huenemoerder, J.H. Kastner, H.L. Marshall, N.S. Schulz, M.W. Wise, J.J. Drake, J.Z. Juda, M. Juda, A.C. Brinkman, C.J.Th. Gunsing, and J.S. Kaastra, G. Hartner and P. Predehl, "Towards the Calibration of the HETG LRF", *Grazing Incidence and Multilayer X-Ray Optical Systems, Proc. SPIE*, Vol. **3113**, (1997).

S.S. Murray and J.H. Chappell, "The Advanced X-ray Astrophysics Facility High Resolution Camera", in *X-ray Instrumentation in Astronomy, Proc. SPIE*, vol. 597, pp. 274-281, 1985.

S.S. Murray, *et al.*, *Astro. Lett. and Communications*, Vol. 26, p. 113, 1987.

P. Predehl, H. Bräuninger, A.C. Brinkman, D. Dewey, J.J. Drake, K.A. Flanagan, C.J.Th. Gunsing, G. Hartner, J.Z. Juda, M. Juda, J. Kaastra, H.L. Marshall, and D. Swartz, "X-Ray Calibration of the AXAF Low Energy Transmission Grating Spectrometer: Effective Area",*Grazing Incidence and Multilayer X-Ray Optical Systems, Proc. SPIE*, Vol. **3113**,(1997).