The *ChIPS* Reference Manual

CIAO Software Release v3.4

December 2006

http://cxc.harvard.edu/ciao/

Contents

1 Introduction **1**

1.0.1 Hands-on introduction to plotting 1
1.0.2 What commands does ChIPS understand? 1
1.0.3 The command line 2
1.0.4 Undo, Redo, Store, & Restore 3
1.0.5 The Display 3
1.0.6 Automatic updating of the display 4
1.0.7 Setting the axis limits for a plot 4
1.0.8 Using multiple plots 5
1.0.9 The INFO command 5
1.0.10 Moving a drawing area in the window 6
1.0.11 Using ChIPS from S-Lang 7
1.0.12 The curve() function 7
1.0.13 Customizing ChIPS 9
1.0.14 The ChIPS state object 9
1.0.15 Saving temporary files 10
1.0.16 Plotting contours 10
1.0.17 Attribute values 11
1.0.18 The ChIPS resource file 11

1.0.19 Format of the resource file 12

1.0.20 COMMAND-LINE OPTIONS 15

1.0.21 XPA ACCESS POINT AND COMMANDS 15

1.0.22 CHANGES IN CIAO 3.2 16

2 *ChIPS* Commands **17**

2.1 $ 17

2.2 # — % 18

2.3 AXES — AXIS 19

2.4 AXIS — AXES 20

2.5 BATCH 20

2.6 BROWSE — PRISM 22

2.7 C 22

2.8 CLEAR 23

2.9 COLORSYS 23

2.10 CONTOUR 24

2.11 CURVE — PLOT 27

2.11.1 Plotting data from a file 27

2.11.2 Plotting data from S-Lang 28

2.11.3 Changing how the plot looks 28

2.11.4 Plotting Multiple Curves 29

2.11.5 Interrupting a Plot 30

2.11.6 ARGUMENT LIST 34

2.11.7 SCOL values 37

2.12 D 37

2.13 DELETE 38

2.14 DISPLAY 40
2.15 DRAWAREA 40
2.16 ERRS 41
2.17 EXIT — QUIT 43
2.18 FONT 46
2.19 GRIDS 47
2.20 INFO 48
2.21 L 49
2.22 LABEL 50
2.23 LEVELS 52
2.24 LIMITS 53
2.25 LINE 55
2.26 LINEAR 56
2.27 LIST 57
2.28 LN 58
2.29 LOCATION 59
2.30 LOG 60
2.31 PACK 61
2.32 PAGESIZE 62
2.33 PICKPOINTS 64
2.34 PLOT — CURVE 65
2.35 PRINT 65
2.36 PRISM — BROWSE 66
2.37 QUIT — EXIT 66
2.38 REDO 66
2.39 REDRAW 67

2.40 RELATIVESIZE — RELSIZE . . . 68
2.41 RESTORE . . . 69
2.42 SKIP . . . 69
2.43 SPLIT . . . 69
2.44 STORE . . . 72
2.45 SURFACE . . . 74
2.46 SYMBOL . . . 76
2.47 TICKS . . . 77
2.48 TICKVALS . . . 80
2.49 TITLE . . . 81
2.50 TYPE . . . 82
2.51 UNDO . . . 82
2.52 VERBOSE . . . 83
2.53 VIEWPOINT . . . 83
2.54 XLABEL . . . 85
2.55 YLABEL . . . 86
2.56 ZLABEL . . . 87

3 S-lang in *ChIPS* **91**

3.1 chips_auto_redraw . . . 91
3.2 chips_clear . . . 92
3.3 chips_color_name . . . 92
3.4 chips_color_value . . . 94
3.5 chips_get_pane . . . 95
3.6 chips_get_xrange . . . 95
3.7 chips_get_xscale . . . 96

3.8 chips_get_yrange 97

3.9 chips_get_yscale 97

3.10 chips_get_zrange 98

3.11 chips_get_zscale 99

3.12 chips_label 99

3.12.1 Quoting the text 100

3.13 chips_line 101

3.14 chips_pickpoints 102

3.15 chips_redraw 106

3.16 chips_set_pane 106

3.17 chips_set_xrange 107

3.18 chips_set_xscale 108

3.19 chips_set_yrange 109

3.20 chips_set_yscale 110

3.21 chips_set_zrange 111

3.22 chips_set_zscale 112

3.23 chips_split 113

3.24 chips_version 114

List of Figures

2.1 <**axesstyle**> Options 21

2.2 <**errortype**> Options 44

2.3 <**errorstyle**> Options 45

2.4 <**symbolstyle**> Options 78

List of Tables

1.2 *ChIPS* State Object Attribute Names and Values 11

3.1 Arguments for chips_auto_redraw 91

3.2 Arguments for chips_color_name 92

3.3 Arguments for chips_color_value 94

3.4 Arguments for chips_label 100

3.5 Arguments for chips_line 101

3.6 Arguments for chips_set_pane 107

3.7 Arguments for chips_set_xrange 107

3.8 Arguments for chips_set_xscale 108

3.9 Arguments for chips_set_yrange 109

3.10 Arguments for chips_set_yscale 110

3.11 Arguments for chips_set_zrange 111

3.12 Arguments for chips_set_zscale 112

3.13 Arguments for chips_split 113

Chapter 1

Introduction

Introduction to *ChIPS*, CIAO's plotting package.

ChIPS (Chandra Imaging and Plotting System) is the plotting package of CIAO and can be used interactively or be driven by a script to create plots on the screen and in hardcopy form. It is embedded within *Sherpa*, is used by applications such as *Prism* and *Filtwin* for displaying data, and can be used to create plots from *S-Lang*.

ChIPS is designed so that plots can be built up interactively in small steps and can easily be refined, saved, printed, and restored. It is particularly flexible at creating plots from tables, but also allows users to generate plots of selected columns from an image or event file. The *S-Lang* programming language can be used to manipulate - or create - data for plotting.

1.0.1 Hands-on introduction to plotting

The CIAO thread "Introduction to Plotting Using *ChIPS*" provides a short, hands-on introduction to using the plotting package. The thread is online at http://cxc.harvard.edu/ciao/threads/chips_intro/ .

1.0.2 What commands does ChIPS understand?

As will be discussed below, *ChIPS* can be controlled from an interactive prompt, from a command file, via XPA, or directly from *S-Lang*. For all but the last case the allowed syntax is given by the following list:

- A *ChIPS* command, such as `EXIT` or `CONTOUR`. Note that *ChIPS* commands are case insensitive, and are generally listed in upper case in the documentation for clarity.

- `ahelp` and `about` can be used to access the CIAO on-line help system. By default any help queries will be restricted to the "chips" context, so "`ahelp print`" will bring up the *ChIPS* command whilst "`ahelp varmm print`" is required to access the Varmm print command.

- A one-line *S-Lang* statement that does not need to end with a semi-colon and in which new variables do not need to be pre-declared. This allows you to enter "`a = 23`" rather than "`variable a = 23;`". See the "Using S-Lang" section below for further details.
- Any line beginning with "$" or "!" is passed through to the shell (after removing this character). This allows you to run shell commands from the *ChIPS* prompt.
- The shell commands "`pwd`", "`ls`", and "`cd`" are available directly from *ChIPS* (so you do not need to say "`!pwd`").
- Any line beginning with the "#" character is ignored.

The *ChIPS* command language - not including the *S-Lang* functions - provides the ability to create and manipulate plots but no data-manipulation capabilities, other than to read in data from a file. The *S-Lang* programming language provides the functions needed to create and manipulate data and to interact with *ChIPS*.

1.0.3 The command line

ChIPS may be launched by typing `chips` on the command line. This brings up the following welcome message -

```
Welcome to ChIPS, version CIAO3.1
Copyright (C) 1999-2003, Smithsonian Astrophysical Observatory

chips>
```

- and leaves you at an interactive prompt at which you can enter any valid *ChIPS*, as discussed above.

There are two ways to start *ChIPS* and get it to execute a pre-determined set of commands. If the ASCII file `plot.chp` contains text that could be entered at the *ChIPS* prompt then it can be processed by saying:

```
unix% chips plot.chp
```

This will cause *ChIPS* to evaluate all the commands in `plot.chp` and then leave you at the interactive prompt. If you want *ChIPS* to exit after processing the commands either add a `QUIT` statement to the end of the file or use the "`--batch`" command-line option and say

```
unix% chips --batch plot.chp
```

New to CIAO 3.0 is the ability to use the store files - created with the `STORE` command - in this way.

If you want to evaluate a *S-Lang* script - which can define useful functions or run through a set of commands - then you have to use the "`--slscript`" option:

```
unix% chips --slscript plot.sl
```

Here the contents of `plot.sl` must be valid *S-Lang* code: variables must be pre-declared (unless you have set the `_auto_declare` variable to 1 at the start of the script) and statements require the closing semi-colon, but *S-lang* statements can be split across more than one line. After processing the *S-Lang* file the interactive prompt will be displayed (the "`--batch`" option can be used to cause *ChIPS* to exit after processing the files).

Any *ChIPS* resource file will be loaded *before* any command-line file (either *ChIPS* or *S-Lang* format) is loaded.

You can load in both *ChIPS* and *S-Lang* files at the same time using

```
unix% chips --slscript plot.sl plot.chp
```

This evaluates the file `plot.sl` and then the file `plot.chp`. Multiple *S-Lang* files can be specified using "`--slscript`" but only one *ChIPS* command file. In fact, the command-line processing stops after the first *ChIPS* command file is found.

Note that the files can be called anything you like: we use the convention that the extensions "`chp`" and "`sl`" indicate *ChIPS* and *S-Lang* files respectively.

1.0.4 Undo, Redo, Store, & Restore

ChIPS records the steps taken to create the current display. This allows you to easily undo recent changes and to restore anything you have just undone using the `UNDO` and `REDO` commands.

It also allows you to save the current plot to a file using the `STORE` command; the resulting file can be loaded into a new *ChIPS* session using the `RESTORE` command. This provides easy editing of plots across multiple runs of *ChIPS* and the ability to easily send actual plots - including the data - to colleagues rather than just a postscript version.

1.0.5 The Display

The primary display device for *ChIPS* is an X-windows display. This will be created the first time a command is issued that modifies the plot display. Once a plot has been created it can be written to a postscript file using the *ChIPS* `PRINT` command.

For those occasions when no X-window output is required - for instance when running a script in the background - "batch" mode can be switched on. If a script calls "`BATCH ON`" or *ChIPS* is started with the `--batch` command-line option

1.0.6 Automatic updating of the display

The default mode for *ChIPS* - when not run in batch mode - is for the plot to be redrawn whenever a command is issued that changes the plot. For instance, after each line of the following is entered the plot will be redrawn:

```
chips> clear
chips> limits 0 10 0.01 10
chips> log y
chips> ylabel "The Y axis"
chips> ylabel size 1.5
chips> ylabel green
```

This is often the desired behaviour, but it can cause problems if the plot contains many data points (since it will take a long time to re-draw the plot) or if the flickering of the contents of the plot window is a nuisance.

The automatic redrawing can be turned off by calling

before creating the plot and then calling

once the plot has finished. If you wish to keep the redraw mode turned off but want to update the window with the current plot contents then call

Note that *Sherpa* sets the redraw mode to OFF when started (or its *S-Lang* module is loaded). The *S-Lang* routines `chips_redraw()` and `chips_auto_redraw()` can be used to set and find the current value of the redraw mode.

So,

```
chips> redraw off
chips> clear
chips> limits 0 10 0.01 10
chips> log y
chips> ylabel "The Y axis"
chips> ylabel size 1.5
chips> ylabel green
chips> redraw on
```

creates the same plot as above but without the plot window being updated until the "`REDRAW ON`" command is called.

1.0.7 Setting the axis limits for a plot

When started, *ChIPS* assumes a single plot with limits of 0-1 along each axis. This is also the state *ChIPS* will be set to after calling either `CLEAR` or `chips_clear`. The `LIMITS` command can be used to change the axis limits, otherwise they will be automatically scaled to match the plotted data. When this automatic mode is used - which is the default behaviour - a gap of 5 per cent of the distance between the minimum and maximum values along the axis is added to each side to improve legibility. This value can be changed (or turned off) by setting the `padfactor` and `fullautolimits` fields of the *ChIPS* state object (described in detail below).

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

The S-Lang functions `chips_get_xrange()` and `chips_set_xrange()` can be used to set and get the X-axis limits for a plot. Similarly named functions exist for the Y and Z axes.

1.0.8 Using multiple plots

The `SPLIT` command is used to split the display window into multiple plots, as described in "`ahelp split`". Each "sub-plot" is referred to as a "Drawing Area" (or "pane") and can contain separate data, axis limits, labels, or any other plot option. The only exception is that there is only one title - you can not have individual titles for each drawing area.

If `SPLIT` is called and a plot (or even multiple plots) already exist then the contents will remain, as long as the total number of plots in the `SPLIT` call is at least as large as the existing number of plots. This is shown in the following example, where the first plot remains even after 5 new panes are added to the display.

```
chips> clear
chips> limits 0 10 10 20
chips> axes red
chips> ylabel "y axis 1"
chips> ylabel size 1.5
chips> split 2 3
```

When *ChIPS* commands are entered they are assumed to be for the current "working" pane. This can be found out by using the `INFO` command, which lists details of all the drawing panels and their contents, as well as indicating the currently selected drawing area. The "`D`" command or *S-Lang* function "`chips_set_pane()`" can be used to switch to different drawing areas (also called panes). The *S-Lang* function "`chips_get_pane()`" can be used to find out what the currently selected pane is.

1.0.9 The INFO command

The `INFO` command lists to the screen information about the current plot settings. As an example:

```
chips> clear
chips> split 3 1
chips> d 2 axes red
chips> info
```

```
Drawing Area  #1
(Location: 0.15 0.4 0.1 0.9)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: defaultcolor  width: 1)

Drawing Area  #2  << CURRENT DRAWING AREA
(Location: 0.4 0.65 0.1 0.9)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: red  width: 1)

Drawing Area  #3
(Location: 0.65 0.9 0.1 0.9)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: defaultcolor  width: 1)
```

Here this shows that there are three drawing areas (or panes) and that the second one is the currently-selected area (since we explicitly prefaced the call to `AXES` with "`D 2`"). The output also lists information such as the axes limits and the location of each pane in the window. Although not shown here, the `INFO` command will also list plotted datasets and lines - again indicating the currently selected curve. Some of this information can be accessed from *S-Lang*; see "`ahelp /chips_get/`" and "`ahelp /chips_set/`" for a list of these functions.

1.0.10 Moving a drawing area in the window

Each drawing area has a location within the display window. By default these do not overlap, but you can use the `LOCATION` command to change the position to embed one plot within another or make one plot cover a larger area than another. The "`INFO`" command gives the values of the current location and the `LOCATION` command is used to change this value. The `DRAWAREA` command is similar to `LOCATION` except that it creates a new drawing area and positions that pane.

```
chips> clear
chips> info

Drawing Area  #1  << CURRENT DRAWING AREA
(Location: 0.15 0.9 0.1 0.9)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: defaultcolor  width: 1)

chips> location 0.15 0.9 0.35 0.9
chips> drawarea 0.15 0.9 0.1 0.3
chips> info

Drawing Area  #1
(Location: 0.15 0.9 0.35 0.9)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: defaultcolor  width: 1)

Drawing Area  #2  << CURRENT DRAWING AREA
```

```
(Location: 0.15 0.9 0.1 0.3)
(Limits  : -0.05 1.05 -0.05 1.05)
(Axes    :  fouraxes   color: defaultcolor  width: 1)
```

In this example we used `LOCATION` to re-size the original drawing area to cover roughly 2/3 of the window (along the Y dimension) and then the `DRAWAREA` command to create a much smaller window below it. Drawing areas can even overlap as shown by the following example (we use the `SPLIT` command to create 3 panes and then resize them, alternatively the second and third panes could have been created using the `DRAWAREA` command):

```
chips> clear
chips> split 1 3
chips> location 0.15 0.9 0.35 0.9
chips> d 2 location 0.15 0.9 0.1 0.3
chips> d 3 location 0.4 0.7 0.2 0.4
```

1.0.11 Using ChIPS from S-Lang

To make *ChIPS* available to a *S-Lang* script you need to load the "`chips`" module by saying:

```
require("chips");
```

This is not necessary if you have already loaded the "`sherpa`" module. A side effect of loading *ChIPS* is that the `Varmm` library is also loaded and the *ChIPS* resource file is evaluated, if it exists.

Once the "`chips`" module is loaded then the *S-Lang* functions - such as `chips_clear()` - can be used. Use

to list these functions. The `chips_eval()` function can be used to run *ChIPS* commands which do not have a *S-Lang* version.

1.0.12 The curve() function

The one exception to the rule that the *S-Lang* function names begin with "`chips_`" is the `curve()` function, which plots the supplied vectors as the `CURVE` command does and can be called as:

```
() = curve( x, y );
() = curve( x, y, yerr );
() = curve( x, y, yerrlo, yerrhi );
() = curve( x, y, xerrlo, xerrhi, yerrlo, yerrhi );
```

As an example, the following *S-Lang* code uses a mixture of *ChIPS* and *S-Lang* commands to create a plot of `sin(x)`:

```
% ensure ChIPS commands are available
require("chips");
% create the data to be plotted
variable x = [1:10:0.02];
variable y = sin(x);
% ensure plot will not be created until the "redraw" call
() = chips_auto_redraw(0);
% set up the limits of the plot
() = chips_set_xrange( 0.5, 10.5 );
() = chips_set_yrange( -1.2, 1.2 );
% draw the data as a line connecting the points rather
% than the default setting (crosses)
chips.symbolstyle = _chips->none;
chips.curvestyle = _chips->simpleline;
() = curve( x, y );
% set up the axis labels
() = chips_eval("xlabel 'X Axis'");
() = chips_eval("ylabel 'Y Axis'");
() = chips_eval("xlabel size 1.5");
() = chips_eval("ylabel size 1.5");
% create the plot
chips_redraw();
```

If this code is saved to a file called "`plot1.sl`" then it can be evaluated by:

```
unix% slsh plot1.sl
```

Whilst it does create the plot, the window is destroyed immediately since the S-Lang script finishes after the call to `chips_redraw()`.

A simple solution is to add code that extends the life of the program - perhaps

```
sleep(30);
```

to make the program last for 30 seconds longer, or

```
variable buffer;
() = printf( "Enter return to exit the program: " );
() = fgets( &buffer, stdin );
```

which will wait until the user hits the return key.

An alternate solution is to create a hardcopy version of the plot rather than an X-windows display. This can be achieved by replacing the call to `chips_redraw()` by

which will create a postscript file called `plot1.ps`.

1.0.13 Customizing ChIPS

ChIPS can be customised by use of the *ChIPS* state object (also called customization variable) and the *ChIPS* resource file.

1.0.14 The ChIPS state object

Several new fields were added to the state object for CIAO 3.0. This object is used to set the defaults for plot colors, styles, and fonts, among other things, and defines the location of temporary files. The contents can be viewed using the Varmm "print()" function:

```
chips> set_state_defaults("chips")
chips> print(chips)
curvestyle       =  2
symbolstyle      =  2
linestyle        =  6
curvecolor       =  3
symbolcolor      =  3
linecolor        =  3
linewidth        =  1
symbolsize       =  3
font             =  3
undo             =  1
savevars         =  1
colorsys         =  rgb
pagewidth        =  8
pagelen          =  8
unit             =  inch
tmpdir           =  /tmp
tmpdata          =  /tmp/.chips.data.username.fits
padfactor        =  0.05
fullautolimits   =  0
lowerloglimit    =  0.01
mingridsize      =  50
```

Many of the fields are direct counterparts to the plot attributes set using ChIPS commands such as curve, plot, symbol, and line. The following table lists the field, a description, and the default value:

Field Name	Description	Default
curvestyle	curve style	_chips->noline
symbolstyle	symbol style	_chips->cross
linestyle	line style	_chips->solid
curvecolor	color of curve	_chips->defcolor
symbolcolor	color of symbols	_chips->defcolor
linecolor	color of lines	_chips->defcolor
linewidth	width of line	1.0
symbolsize	size of symbols	3.0

font	font used for text	_chips->roman
undo	turns undo on/off	1 (*i.e.* on)
savevars	see "Saving temporary files" section	1 (*i.e.* on)
colorsys	color system for plotting	rgb
pagewidth	x dimension of the hardcopy plot	8.0
pagelen	y dimension of the hardcopy plot	8.0
unit	unit for pagewidth and pagelen	inch
tmpdir	location of temporary files (see "Saving temporary files" section)	$ASCDS_WORK_PATH
tmpdata	name of temporary files (see "Saving temporary files" section)	<chips.tmpdir>/.chips.data.$LOGNAME.fits
padfactor	padding for plots	0.05
fullautolimits	if on, includes labels and lines in calculating auto limits	0 (*i.e.* off)
lowerloglimit	lower limit threshold for log scales	0.01
mingridsize	size of the interpolated data grid (see "Plotting contours" section)	50

The allowable options for the relevant fields are discussed in the "Attribute values" section.

1.0.15 Saving temporary files

Three fields in the state object are related to the behavior of ChIPS when saving temporary data: savevars, tmpdir, and tmpdata. If chips.savevars is set to 1, then plots generated from S-Lang variables will be saved to a temporary file during undo and redo operations. It should be noted that setting this variable to 1 can make the tool very slow when redrawing either large plots or many small ones. The location and name of the temporary file is controlled by the tmpdir and tmpdata parameters, respectively.

1.0.16 Plotting contours

The chips.mingridsize state object field has been added to CIAO 3.0 to improve contour plotting for cases where the data is not uniformly spaced. This variable is used to interpolate the data used in creating the regular grid required by the underlying plotting engine. The default value of 50 indicates a 50x50 grid; a higher value means finer resolution of interpolated data and therefore more accuracy in the calculated contour.

If data is uniform, chips.mingridsize is not used in linear scale. In log scale, it is used only if the data grid supplied by user is smaller than the current chips.mingridsize setting; this determination is done independently for X and Y coordinates.

The value of chips.mingridsize may be set at the prompt. After doing so, change from linear to log scale and back (or vice-versa) to recalculate the grid. Note that the appearance of the contours may not be accurate because of the interpolation.

1.0.17 Attribute values

Each distinct ChIPS attribute (*e.g.* symbol, color, font) is reflected as a constant within the '_chips' namespace. For example, to set the symbol color attribute to use RED you would say

(the trailing semi-colon is only needed if used in *S-Lang* code rather than entered at the *ChIPS* prompt).

Attribute type	Attribute values
Color	_chips->black, _chips->blue, _chips->cyan, _chips->defcolor, _chips->green, _chips->magenta, _chips->red, _chips->white, _chips->yellow
Curve Styles	_chips->histo, _chips->noline, _chips->simpleline, _chips->step
Line Styles	_chips->dash, _chips->dashlongdash, _chips->dot, _chips->dotdash, _chips->dotlongdash, _chips->longdash, _chips->solid
Symbol Styles	_chips->bigpoint, _chips->block, _chips->circle, _chips->cross, _chips->diamond, _chips->downtri, _chips->none, _chips->point, _chips->soliddiamond, _chips->soliddowntri, _chips->soliduptri, _chips->square, _chips->uptri
Font Names	_chips->greek, _chips->italic, _chips->oldenglish, _chips->roman, _chips->tiny
Color Systems	_chips->cmyk, _chips->grayscale, _chips->rgb
Units	_chips->cm, _chips->inch, _chips->mm

Table 1.2: *ChIPS* State Object Attribute Names and Values

For the color values, the S-Lang functions `chips_color_name()` and `chips_color_value()` are also available to help convert between text (*e.g.* "RED") and numeric (*e.g.* 6) forms.

It is important to note that the *ChIPS* state object is currently not saved by the `STORE` command. So, after using the `RESTORE` command, data will be drawn correctly but the values of the *ChIPS* state object will not be changed.

1.0.18 The ChIPS resource file

When a *ChIPS* session is started - either directly by the "chips" command or from another CIAO application such as *Sherpa*, `prism` or a *S-Lang* program - it looks for a *ChIPS* resource file. If found, the contents are processed at the start of the session.

The resource file must be in one of the following locations:

- the $CHIPSRC environment variable
- $PWD/.chipsrc
- $HOME/.chipsrc

The search stops when the first match is made and *ChIPS* is launched, even if the chosen resource file contains an error.

The following is an example $HOME/.chipsrc file that causes any application that starts *ChIPS* to print two messages and set a number of plot defaults:

```
message("Starting to process .chipsrc")
chips.curvecolor  = _chips->blue
chips.symbolcolor = _chips->red
chips.linecolor   = _chips->green
chips.curvestyle  = _chips->histo
chips.symbolstyle = _chips->diamond
chips.font        = _chips->italic
message("Finished processing .chipsrc")
```

After this file is read, any columns plotted from prism, for example, will consist of red diamonds connected by a blue line using the histogram (step) mode, and the axis labels will be italic.

Although the first and last lines of the above example create screen output for demonstrative purposes, it is recommended that the .chipsrc file does not contain any command that creates either text or graphical output.

1.0.19 Format of the resource file

Since the resource file is a *ChIPS* (not *S-Lang*) script, it may contain simple, one-line *S-Lang* statements; this is in contrast to the Varmm resource file, which can contain any set of valid *S-Lang* statements. The "evalfile" command can be used to embed *S-Lang* function definitions, or other statements that require more than one line, into the resource file.

So, if `/home/ciaouser/slang/setup_chips.sl` contained

```
% set up the fields of the ChIPS
% - this is a S-Lang file so all statements need to end in ;
chips.curvecolor  = _chips->blue;
chips.symbolcolor = _chips->red;
chips.linecolor   = _chips->green;
chips.curvestyle  = _chips->histo;
chips.symbolstyle = _chips->diamond;
chips.font        = _chips->italic;

% define some useful functions
define xaxis(label) {
  variable oldval = chips_auto_redraw(0);
  () = chips_eval( "xlabel '" + string(label) + "'" );
  () = chips_eval( "xlabel size 1.5" );
  () = chips_auto_redraw(oldval);
}
```

```
define yaxis(label) {
  variable oldval = chips_auto_redraw(0);
  () = chips_eval( "ylabel '" + string(label) + "'" );
  () = chips_eval( "ylabel size 1.5" );
  () = chips_auto_redraw(oldval);
}
```

then this could be made available to *ChIPS* by adding the line

to your .chipsrc file. Note that we use the full path to the file so it can be found wherever *ChIPS* is started from. See "**ahelp chips_eval**" for more details on the "**xaxis()**" function.

For a full list of *ChIPS* commands and *S-Lang* functions use "ahelp -c chips".

Examples:

1. **Starting ChIPS**

   ChIPS can be started at the command line by the "**chips**" command. This will start up *ChIPS*, evaluate any commands in the resource file, and leave you at the interactive prompt.

   ```
   unix% chips

   Welcome to ChIPS, version CIAO3.1
   Copyright (C) 1999-2003, Smithsonian Astrophysical Observatory

   chips> info

   Drawing Area  #1  << CURRENT DRAWING AREA
   (Location: 0.15 0.9 0.1 0.9)
   (Limits  : -0.05 1.05 -0.05 1.05)
   (Axes    :  fouraxes   color: defaultcolor  width: 1)

   chips>
   ```

   Here we used the `INFO` command to display the current plot settings. As no plots have been created it lists the default settings.

2. `chips> curve "lc.fits[cols time,rate]"`
   `chips> red`

   Plotting two columns from a FITS file

   The `CURVE` command is used here to plot the RATE column against the TIME column from the file `lc.fits`. The column selection is made using the Data Model syntax (see "**ahelp dmcols**"); in fact any valid Data Model "virtual file" syntax ("**ahelp dmsyntax**") can be used to filter the file before it is displayed.

   Once the data is displayed the look of the plot can be manipulated using commands such as `RED`. See the following example for further details.

3. **Changing the way a curve looks**

There are two ways of changing the presentation of a *ChIPS* plot. First, we use the *ChIPS* state object to create a plot consisting of white crosses connected by a red line:

```
chips> x = [1:10:0.1]
chips> y = sin(x)
chips> chips.curvecolor = _chips->red
chips> chips.curvestyle = _chips->simpleline
chips> curve( x, y )
0
```

The "0" after the curve command indicates that it was a success; it can be avoided by saying "`() = curve(...)`" as shown below.

This plot may also be created using *ChIPS* commands to alter the attributes of the curve:

```
chips> set_state_defaults("chips")
chips> clear
chips> () = curve( x, y )
chips> simpleline
chips> red
```

The main difference between the two approaches is that values in the *ChIPS* state object are persistent, meaning that they will be used as the defaults for any new plotting objects that are subsequently created, whilst the *ChIPS* commands like `RED` only refer to the current plot item.

These examples would produce the same results if the *ChIPS* CURVE command had been used - *i.e.* "`curve x x y`" - rather than the *S-Lang* "curve()" function.

4. **Calling ChIPS commands from S-Lang**

In CIAO 3.0 a number of *S-Lang* functions have been added to set and return information about plots (See "`ahelp /chips_/`" for a list of them). However, many of the *ChIPS* commands do not have a *S-Lang* version and must be called via `chips_eval()`. The following two lines show how a *S-Lang* script would - after a 'require("chips");' statement - be able to set the title of a plot.

```
() = chips_eval( "title 'Plot title'" );
() = chips_eval( "title size 1.5" );
```

See "`ahelp chips_eval`" for more information on this.

5. **Temporarily disabling the use of a resource file**

If you wish to temporarily disable the processing of the *ChIPS* resource file, use the following c-shell or ksh command sequences to make *ChIPS* read from an empty or non-existent file; for c-shell:

```
unix% ( setenv CHIPSRC /dev/null; chips )
```

and for ksh:

```
unix% ( export CHIPSRC=/dev/null; chips )
```

1.0.20 COMMAND-LINE OPTIONS

The command-line options for *ChIPS* - which can also be listed by entering "**chips -help**" - are:

Option	Description
`--batch`	Runs in batch mode: the *ChIPS* welcome message is not displayed, any supplied code is run as if "`BATCH ON`" were specified, and *ChIPS* exits after evalulating the code.
`-h`, `-help`, or `--help`	Lists the command-line options.
`-nosession`	Do not attatch this *ChIPS* to the current *CIAO* session.
`--slscript` <`filename`>	Evaluates the *S-Lang* code in <`filename`>.
`-xpa` <`name`>	Sets the XPA access point of *ChIPS* to <`name`>.

See the section titled "The command line" in the main discussion for more details on how *ChIPS* and *S-Lang* code can be automatically executed by *ChIPS* using these options.

1.0.21 XPA ACCESS POINT AND COMMANDS

ChIPS has an XPA access point which can be used to control its behaviour. This is used by the CIAO session handling ("ahelp session") to provide an integrated analysis environment.

The default name is "`chips`", with further instances being called "`chips2`", "`chips3`", ...(the "`-xpa`" command-line option can be used to override this naming scheme). *ChIPS* understands the following commands when sent via XPA:

```
unix% xpaget prism
cmd:            Any valid ChIPS command
                options: see ChIPS documentation for each command
exit:           Exit the current application
                options: None
quit:           Synonym for "exit"
```

If you have a *ChIPS* program running then the following will clear its display, set the plot limits to 0-10 on both axes, and draw a line between (1,1) and (8,9):

```
unix% xpaaccess chips
yes
unix% xpaset -p chips "cmd clear"
unix% xpaset -p chips "cmd limits 0 10 0 10"
unix% xpaset -p chips "cmd line 1 1 8 9"
```

You can also send in any *S-Lang* command you can enter at the *ChIPS* prompt. For example:

```
unix% xpaset -p chips "cmd () = chips_set_xrange(0,9)"
```

The '`() =`' is needed to avoid the return value being printed in the *ChIPS* terminal window.

It is not possible to use the **`xpaget`** command to get information from *ChIPS*. Sending a command like "`INFO`" via XPA results in the output being printed to the *ChIPS* terminal window.

1.0.22 CHANGES IN CIAO 3.2

The ChIPS module can now be loaded by using the `require(‘‘chips");` statement, although the previous method (loading with the import command) still works.

Chapter 2

ChIPS Commands

All of the information provided here is available within CIAO and *ChIPS* from the **ahelp** system; "**ahelp -c chips**" returns a list of all the help files relating to *ChIPS* .

ChIPS also understands the S-Lang programming language; see Chapter 3 for more information.

The following conventions are used for describing the syntax:

- `chips>` is the *ChIPS* command-prompt.
- `<arg>` is an argument to a command.
- `#` is an integer argument.
- Brackets - [ ] - indicate an optional argument.
- A pipe - | - means "or". By extension, {`<command_a>` | `<command_b>` } indicates a choice between equivalent commands and {`<arg_a>` | `<arg_b>` } indicates a choice between different arguments.

Keep in mind that *ChIPS* commands are **not case sensitive** and so may be issued in either uppercase or lowercase.

2.1 $

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.

2.2 # — %

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
```

2.3 AXES — AXIS

Specifies the appearance of the bounding box surrounding a drawing area. The command AXIS is equivalent.

```
chips> [D #] [{AXES | AXIS}] <axesstyle>
chips> [D #] {AXES | AXIS} <color>
chips> [D #] [{AXES | AXIS}] <scale> [<coor>]
chips> [D #] {AXES | AXIS} WIDTH <width_value>
```

```
Argument: D #
Description: drawing area number designation
Options: integer number
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <axesstyle>
Description: axes style
Options: FOURAXES - all four axes, with tick marks
         TWOAXES - left and bottom axes only, tick marks on both
         TWOAXESFRAME - left and bottom axes with tick marks,
                        right and top axes without ticks
Default: FOURAXES
```

```
Argument: <color>
Description: axes color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE, YELLOW
Default: DEFAULT (appears in ChIPS window as white, prints as black)
```

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: both X and Y
```

```
Argument: <scale>
Description: axis scale
Options: LINEAR, LOG
Default: LINEAR
```

```
Argument: <width_value>
Description: axes thickness
Options: real number
Default: 1.0
```

For contour and surface plots, the <scale> cannot be changed; additionally, <axesstyle> cannot be changed for surface plots. See the `CONTOUR` (Section 2.10) and `SURFACE` (Section 2.45) commands for further information about plotting 2-D data in *ChIPS*.

See the `TICKS` (Section 2.47) and `TICKVALS` (Section 2.48) commands for information on specifying other attributes of the box surrounding the drawing area.

Figure 2.1 is an illustration of the different <`axesstyle`> options.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**`ahelp chips`**" for further details.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> AXES CYAN
   chips> AXES WIDTH 4.0
   chips> AXES TWOAXES
   chips> LOG
   ```
   First, a plot is produced using the CURVE command. The subsequent AXES commands change the axes to the color CYAN, make the lines composing the axes thicker (to the value of 4.0), and change the style to include only the left and bottom axes. The final command switches both the X and Y axes to logarithmic scale.

2.4 AXIS — AXES

Specifies the appearance of the bounding box surrounding a drawing area.

See the `AXES` command (Section 2.3), for further information and examples.

2.5 BATCH

Turns on/off use of a plotting device.
   ```

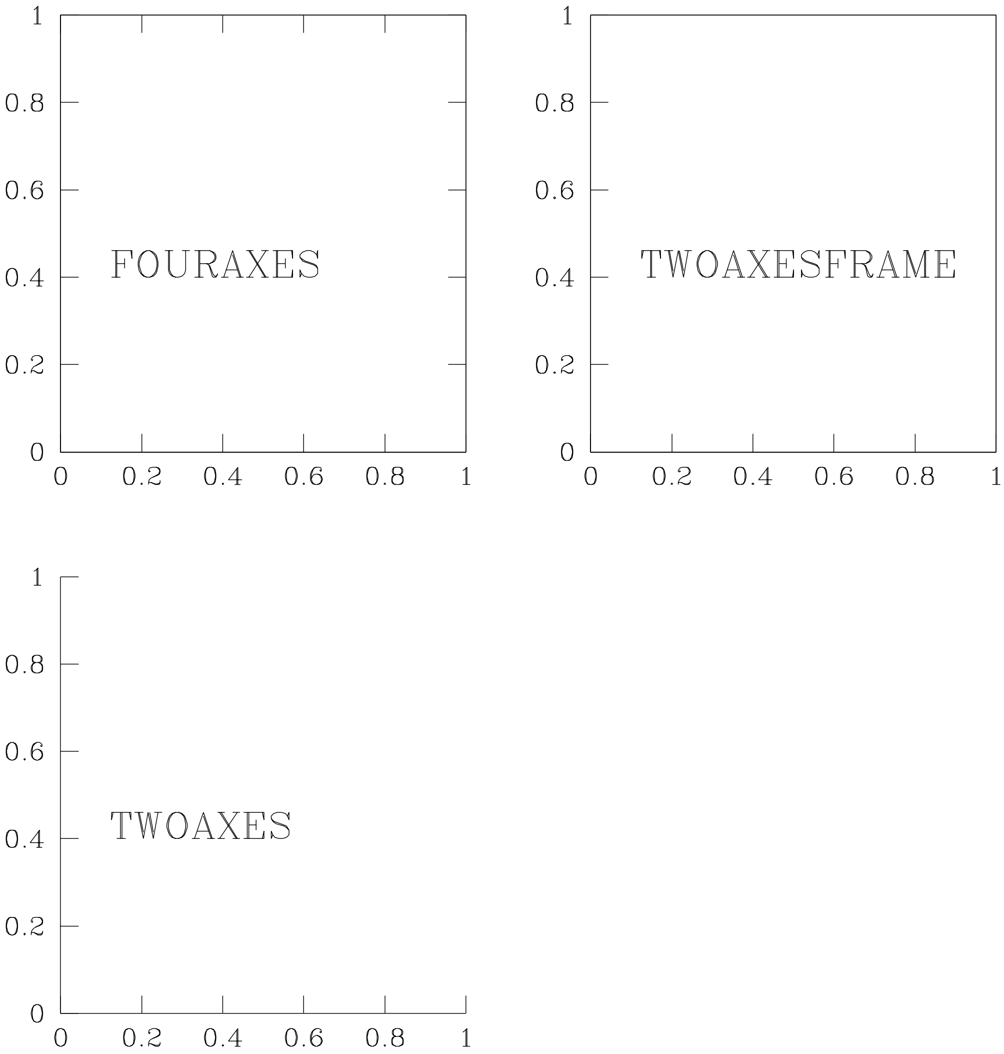

Figure 2.1: <**`axesstyle`**> Options

```
chips> BATCH {ON | OFF}
```

By default, BATCH is turned off (use of a plotting device is turned on).

Examples:

1. ```
   chips> BATCH ON
   chips> CURVE data/example.dat
   chips> PRINT POSTFILE batch.myplot.ps
   ```

   By turning on the BATCH command, use of a plotting device is turned off. This means that when the CURVE command is issued, the plotting device is not opened. The results of the CURVE command (and any other *ChIPS* commands that are issued) may still be printed to a PostScript file using the `PRINT` command (Section 2.35).

2.6 BROWSE — PRISM

Launches the CIAO file-browsing GUI, *Prism*. The command PRISM is equivalent.

```
chips> {BROWSE | PRISM} [<filename>]
```

If a *Prism* process is already running, issuing this command with the <filename> specification will utilize the current *Prism* process. Otherwise, a new *Prism* process will be invoked.

The *Prism* GUI will remain open after exiting *ChIPS*.

2.7 C

Defines the current curve.

```
chips> [D #] C {# [,#, ...] | #:# | ALL}
```

where # is an integer argument, #:# specifies an inclusive range of curve numbers, and D # may specify the number of the drawing area.

See the `D` command (Section 2.12) for more information about this argument.

Note that wherever the "C #" is used, the above extended syntax may be utilized to alter multiple curves at once. If a non-existent curve is specified, *ChIPS* will act on the current curve.

Examples:
```

1. `chips> CURVE data/exampleA.dat`
   `chips> CURVE data/exampleB.dat`
   `chips> SIMPLELINE`
   `chips> LABEL 6.0 50.0 "Curve 1"`
   `chips> LABEL 6.0 5.0 "Curve 2"`

   The two curves are numbered in the order in which they were plotted. The most recently plotted curve (curve 2) is the current curve. Therefore, `SIMPLELINE` acts on curve 2, then each curve is labeled.

2. `chips> D 1 C 1`
   `chips> SIMPLELINE`

   Building on the prvious example, the command "D 1 C 1" selects curve 1 as the current curve. The subsequent `SIMPLELINE` command acts on curve number 1, changing the style to a line.

3. `chips> C 2`
   `chips> WIDTH 4.0`

   The command C 2 defines curve number 2 as the current curve, then the line is changed to have a width of 4.0.

4. `chips> C 1,2`
   `chips> BLUE`

   Both curves are chosen such that the `BLUE` command acts on both curves. In this example, the following commands would be equivalent to the "C 1,2" command:

   `chips> C 1:2`

   `chips> C ALL`

2.8 CLEAR

Removes all plotting objects and creates a new blank drawing area.

```
chips> CLEAR
```

The newly-created drawing area is given the number designation 1.

Examples:

1. `chips> CLEAR`

   This command clears the plotting window and restores the drawing area number designation to 1.

2.9 COLORSYS

Sets the color system for hardcopy plots.

```
chips> colorsys <color>
```

In this context, “hardcopy” refers to both sending the plot directly to a printer and printing it to a postscript file.

```
Argument: <color>
Description: color system
Options: CMYK, GRAYSCALE, RGB
Default: RGB
```

More information on creating hardcopies is available from “`ahelp chips print`”.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See “`ahelp chips`” for further details.

Unlike most other *ChIPS* commands, the “colorsys” command modifies the state object. This means that once the color system is changed, the command does not need to be reissued. The current color system may also be determined by checking the state object, as shown in the EXAMPLES section.

Examples:

1. ```
   chips> x = [1:10:0.1]
   chips> y = sin(x)
   chips> () = curve( x, y )
   chips> simpleline
   chips> red
   chips> axes yellow
   chips> colorsys grayscale
   chips> print postfile gray.ps
   ```
   A plot is created and a few attributes are modified. The color system is then set to grayscale, and a postscript file is created.

2. ```
   chips> print(chips.colorsys)
   rgb
   chips> colorsys cmyk
   chips> print(chips.colorsys)
   cmyk
   ```
   This illustrates that changing the color system on the command line also updates the *ChIPS* state object field.

2.10 CONTOUR

Adds a contour plot to a drawing area.

```
chips> [D #] CONTOUR <filename>[<virtual_file_syntax>] [LEVELS] <lev1> <lev2> ... <levN>
```

```
Argument: D #
Description: drawing area number designation
Options: integer number
Default: current drawing area
```

See the D command (Section 2.12) for more information about this argument.

```
Argument: <filename>
Description: name of datafile and path, if necessary.  If the file
             contains tabulated data, it must be sorted with row
             precedence (see below).  May be filtered data from a
             FITS image file, as shown in the EXAMPLES.
```

```
Argument: <virtual_file_syntax>
Description: A filtering and/or binning command argument.  The
             <virtual_file_syntax> must be surrounded by
             quotes (" ").
```

See the CURVE command (Section2.11) for more information on this argument.

```
Argument: <lev1> <lev2> ... <levN>
Description: numbers specifying Z axis values of each contour level
Options: real numbers
Description: the <lev1> value specifies the Z axis data value of
             the first contour line, <lev2> specifies the value
             of the second contour line, and so on.
```

See the LEVELS command (Section 2.23) for information on setting the levels in the current contour plot.

Contour plots are considered variants of curves, and therefore the attributes of contours (color, linestyle, scale, width) may be changed in the same manner. However, the limits of the x and y axes in a contour plot cannot be modified.

New functionality for plotting contours was added to *ChIPS* in CIAO 3.0 via the "**chips.mingridsize**" state object field; see the "Plotting contours" section of "**ahelp chips**" for more information.

SORTING THE INPUT DATA

The input data file must be sorted with row precedence. For example, the following data:

```
1 1 1
2 1 -4.97206e-24
3 1 2
4 1 1
5 1 2
1 2 -6.79138e-24
```

```
2 2 -2.23247e-23
3 2 1
4 2 -2.1879e-22
5 2 1
```

Would have to be re-ordered for use in *ChIPS*:

```
1 1 1
1 2 -6.79138e-24
2 1 -4.97206e-24
2 2 -2.23247e-23
3 1 2
3 2 1
4 1 1
4 2 -2.1879e-22
5 1 2
5 2 1
```

Examples:

1. `chips> CONTOUR data/sorted.dat 2 4 20`

   Plots the ASCII data file data/sorted.dat as a contour plot, with the first contour line at a Z-axis value of 2, the second at a value of 4, and the third at a value of 20.

2. `chips> CONTOUR data/exampleImage.fits[10:100,10:100] 2 4 20`

   Plots the data in the specified ranges from the FITS file. The Z-axis values are the same as in Example 1.

3. ```
   chips> C 1 DEL
   You have no more curves in this drawing area.
   chips> CONTOUR "data/sorted.fits[cols col1, col2, col3]" 2 4 20
   chips> C 1 WIDTH 4.0
   ```

   The existing curve is deleted before the data in the first three columns of data/sorted.fits is plotted; the same Z-axis values as before are used. The final command changes the contour curve to a thicker width.

4. `chips> CONTOUR s3_img.fits 10 20 30`

   A contour plot is created from the filtered data stored in "s3_img.fits".

5. ```
   chips> CONTOUR data/data3D.dat 40 50 60
   chips> C 1 DOT
   chips> CONTOUR data/data3D.dat 70 80 90
   chips> C 2 LONGDASH
   ```

   Two contour curves are plotted with different Z-axis values. The linestyle of each is changed so that the levels may be differentiated.

2.11 CURVE — PLOT

Plots a curve or change its attributes (PLOT is equivalent).

```
chips> CURVE <filename> # # #
chips> PLOT <filename> # # #

chips> CURVE <filename> [X #] [Y #] \
[{XERR # | XUP # | XDOWN # | XDN #}] \
[{E # | YERR # | YUP # | YDOWN # | YDN #}] \
[SCOL #]
chips> PLOT <filename> [X #] [Y #] \
[{XERR # | XUP # | XDOWN # | XDN #}] \
[{E # | YERR # | YUP # | YDOWN # | YDN #}] \
[SCOL #]

() = curve( x, y );
() = curve( x, y, yerr );
() = curve( x, y, yerrlo, yerrhi );
() = curve( x, y, xerrlo, xerrhi, yerrlo, yerrhi );
```

When used to plot data from a file CURVE and PLOT are synonyms but curve() is the only valid S-Lang function name.

2.11.1 Plotting data from a file

The CURVE command is used to plot data from either an ASCII data file or a FITS table file.

If you are only plotting x against y values - *i.e.* no errors - then you just need to specify the columns using the Data Model column filtering syntax (see “ahelp dmcols”). So

```
  chips> curve "lc.fits[cols time,rate]"
```

plots the rate column (Y axis) against the time column (X axis). For an ASCII file this would just be

```
chips> curve lc.dat
```

To include error bars you need to specify - after the filename - which columns contain the error information. If only symmetric errors on the Y axis are required then the simplest way is to say

```
chips> curve lc.fits 1 3 5
```

- which means that the X and Y columns are taken from the first and third column of lc.fits and the error on the Y value is taken from the fifth column. This can also be written as

```
chips> curve lc.fits x 1 y 3 e 5
```

where we explicitly include the type for each column. If the X and Y columns are the first two columns in the file then this can be shortened to

```
chips> curve lc.fits e 5
```

(`yerr` can be used instead of `e`).

For non-symmetric Y errors, use the `YUP` and `YDOWN` (or `YDN`) labels to give the the number of the columns containing the low and high error values. The `XERR`, `XUP`, and `XDOWN` (or `XDN`) labels indicate the numbers of the columns which contain errors for the X axis.

The `SCOL` label is used to indicate the column from which the symbol type for each element is found. This column must be an integer and the values are discussed below.

2.11.2 Plotting data from S-Lang

The *S-Lang* function "`curve()`" can be used to plot two arrays against each other. So,

```
variable x = [0:10:0.01];
() = curve( x, sin(x) );
```

draws sin(x) for all points between 0 and 10 with a step-size of 0.01. As shown in the SYNTAX section above, the curve() command can also plot Y errors (symmetric or non-symmetric) and for the X axis too (although if X errors are given then the YERRLO, YERRHI, XERRLO, and XERRHI values must be given). So,

```
variable x = [1:20];
() = curve( x, x+2, sqrt(x+2) );
```

plots up a set of points with symmetric Y errors.

The `curve()` functions returns 0 on success, -1 on error. In the examples above we use "() =" to ignore this return value.

2.11.3 Changing how the plot looks

Whether a curve has been created from a file using the *ChIPS* `CURVE` command, or created from *S-Lang* using the `curve()` function, the same set of commands can be used to change its attributes.

Firstly, all curves are created with their attributes set to the values given in the *ChIPS* state object. Therefore, setting

```
chips.symbolstyle = _chips->square;
```

(whether in the *ChIPS* resource file or at the *ChIPS* prompt) will cause any new curves to be drawn using an open square for the sybol. Note that this will *not* change the attributes of curves that have already been drawn.

Secondly, the `CURVE` command can be used to change the attributes of a curve that has already been drawn. Each of the following commands can be preceeded by one - or both - of [`D #`] and [`C #`] to select the curve you wish to change. See "`ahelp d`" and "`ahelp c`" for more information on these two commands.

```
chips> [{CURVE | PLOT}] <color>
chips> [{CURVE | PLOT}] <curvestyle>
chips> [{CURVE | PLOT}] <linestyle>
chips> [{CURVE | PLOT}] SCALE [<coor>] <scale_value>
chips> [{CURVE | PLOT}] WIDTH <width_value>
chips> ERRS [<coor>] <color>
chips> ERRS [<coor>] <errorstyle>
chips> ERRS [<coor>] <errortype>
chips> ERRS [<coor>] <linestyle>
chips> ERRS [<coor>] {ABSOLUTE | RELATIVE}
chips> ERRS [<coor>] SIZE <errortick_size_value>
chips> SYMBOL <color>
chips> SYMBOL <symbolstyle>
chips> SYMBOL [SIZE] <symbol_size_value>
```

A description of the various arguments used in these calls can be found in the "ARGUMENT LIST" section at the end of this document.

2.11.4 Plotting Multiple Curves

Multiple curves may be plotted within the same drawing area. The limits of the drawing area are affected by the {CURVE | PLOT} command as follows:

- When the first curve is added, the limits of the drawing area are automatically sized to fit.
- When each subsequent curve is added, *ChIPS* automatically changes the limits of the drawing area. If the range and domain of the new curve overlaps with the range and domain of the current drawing area, the drawing area is resized, using as new limits the smallest minimum and the largest maximum values. If either the range or the domain of the new curve do not overlap with the range or domain of the current drawing area, the limits are set to display the new curve, and a warning message is given to indicate that the drawing area limits were changed significantly.

This automatic calculation of the plot limits can be stopped by using the `LIMITS` command to set the axis ranges to fixed values.

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

2.11.5 Interrupting a Plot

Note that *ChIPS* does not recognize "CTRL-C" as an interrupt command. There is no way to cancel the drawing of a long plot once it has begun.

Examples:

1. `chips> CURVE "lc.fits[cols time,rate]"`

   Here we plot the rate versus time columns from the FITS file "lc.fits". The Data model column filter "[`cols time,rate`]" is used to select the X and Y columns for the plot (since we do not explicitly set them they default to the first two columns in the file). The following commands are equivalent:

   ```
   chips> D 1 CURVE "lc.fits[cols time,rate]"
   chips> CURVE "lc.fits[cols time,rate]" X 1 Y 2
   ```

2. ```
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> SYMBOL SQUARE
   ```

   First we remove any existing data from *ChIPS* with the `CLEAR` command, otherwise the new data would be overplotted on any existing data. The data is ploted as in the previous example except that we then change the symbol to be a square.

3. ```
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> SYMBOL SQUARE
   chips> SYMBOL RED
   ```

   This changes the symbol type to a red square.

4. ```
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> SIMPLELINE
   chips> SYMBOL NONE
   ```

   The default behaviour for `CURVE` is to draw symbols at each point. Here we change it to drawing lines between each point and then remove the symbols.

5. ```
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> SIMPLELINE
   chips> RED
   ```

   Here we draw a red line between the points but leave the symbol. Note that the symbol color has not been changed to red: this would be done by saying

   ```
   chips> SYMBOL RED
   ```

6. ```
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> STEP
   chips> RED
   chips> SYMBOL YELLOW
   ```

   There are two ways that points can be connected by lines: `SIMPLELINE` draws lines between the points whereas `STEP` (or its alias `HIST`) considers the data points to represent a histogram (the X values give the center of each bin).

   The line color is changed to red and the symbol color is changed to yellow.

7. ```
   chips> REDRAW OFF
   chips> CLEAR
   chips> CURVE "lc.fits[cols time,rate]"
   chips> STEP
   chips> RED
   chips> SYMBOL YELLOW
   chips> REDRAW ON
   ```

   This has the same result as the previous example except that the plot is not changed until the `REDRAW ON` command is called. See "**ahelp redraw**" for more information.

8. ```
   chips> CLEAR
   chips> chips.curvestyle = _chips->step
   chips> chips.curvecolor = _chips->red
   chips> chips.symbolcolor = _chips->yellow
   chips> CURVE "lc.fits[cols time,rate]"
   ```

   The previous examples all used the `CURVE` command to change the attributes of an existing plot. The *ChIPS* state object (see "**ahelp chips**") provides a way of setting the attributes for any new plots. With the above commands the result is the same plot as previous (a set of points connected as if a histogram with a red line and yellow symbols). The difference is that if another curve is plotted it will also be drawn with the same settings.

9. ```
   chips> chips_clear
   chips> lc = readfile("lc.fits")
   chips> () = curve(lc.time,lc.rate)
   chips> STEP
   chips> RED
   chips> SYMBOL YELLOW
   ```

   Here we have read in the contents of "lc.fits" using the Varmm function **readfile** (see "**ahelp readfile**") and have used the *S-Lang* function `curve()` to plot the time and rate columns of the file. Once the curve has been created it can be manipulated as any other curve.

10. ```
    chips> chips_clear
    chips> chips.curvestyle = _chips->step
    chips> chips.curvecolor = _chips->red
    chips> chips.symbolcolor = _chips->yellow
    chips> lc = readfile("lc.fits")
    chips> () = curve(lc.time,lc.rate)
    ```

    The `curve()` command also follows the settings of the *ChIPS* state object when creating the plot.

11. ```
    chips> CURVE data/exampleB.dat
    ```

Plots the ASCII data file data/exampleB.dat using the default settings, where column 1 is Xx and column 2 is y. The following commands are equivalent:

```
chips> D 1 CURVE data/exampleB.dat
chips> CURVE data/exampleB.dat X 1 Y 2
```

12. `chips> CURVE "data/exampleB.fits[cols col1,col2]"`

    The <virtual_file_syntax> argument is used to plot columns col1 and col2 from a FITS file. Again, the default setting are used, *i.e.* the first column listed in the DM filter (col1) is x and the second (col2) is y. The following commands are equivalent:

    ```
    chips> D 1 CURVE "data/exampleB.fits[cols col1,col2]"
    chips> CURVE "data/exampleB.fits[cols col1,col2]" X 1 Y 2
    ```

13. `chips> CURVE "data/exampleB.fits[cols col1,col2,col6]" 1 2 3`

    Columns named col1, col2, and col6 are plotted from the FITS data file data/exampleB.fits. The first column listed in the DM filter is x, the second is y, and the third is the symmetric error bar in y. The following commands are equivalent:

    ```
    chips> D 1 CURVE "data/exampleB.fits[cols col1,col2,col6]" 1 2 3
    chips> CURVE "data/exampleB.fits[cols col1,col2,col6]" X 1 Y 2 E 3
    chips> CURVE "data/exampleB.fits[cols col1,col2,col6]" X 1 Y 2 YERR 3
    ```

14. `chips> CURVE data/exampleB.dat X 1 Y 2 YUP 7 YDOWN 8`

    The ASCII data file data/exampleB.dat is plotted, with column 1 as x, column 2 as y, column 7 as the positive error in y, and column 8 as the negative error in y. The following commands are equivalent:

    ```
    chips> D 1 CURVE data/exampleB.dat X 1 Y 2 YUP 7 YDOWN 8
    chips> CURVE data/exampleB.dat YUP 7 YDOWN 8
    ```

15. ```
    chips> CURVE "data.fits[cols col1, col2, col7, col8]" X 1 Y 2
           YUP 3 YDOWN 4
    ```

    The same as the previous example, but the data is read from a FITS file. The following commands are equivalent:

    ```
    chips> D 1 CURVE "data.fits[cols col1, col2, col7, col8]" X 1 Y 2
           YUP 3 YDOWN 4
    chips> CURVE "data.fits[cols col1, col2, col7, col8]" YUP 3 YDOWN 4
    ```

16. `chips> CURVE data/exampleB.dat X 1 Y 2 XUP 4 XDOWN 5 YERR 6`

    This command plots x, y, the positive and negative errors in x, and the symmetric error in y. The following commands are equivalent:

    ```
    chips> D 1 CURVE data/exampleB.dat X 1 Y 2 XUP 4 XDOWN 5 YERR 6
    chips> CURVE data/exampleB.dat XUP 4 XDOWN 5 YERR 6
    ```

17. ```
    chips> CURVE data/exampleA.dat
    chips> CURVE data/exampleB.dat
    ```

    Two curves are plotted and numbered 1 and 2, respectively.

18. ```
chips> C 1 DEL
You have no more curves in this drawing area.
chips> CURVE data/exampleB.dat 1 2 6
```
An existing curve is deleted before the ASCII data is plotted; column 1 is x, column 2 is y, column 6 is the symmetric error bar in Y. The following commands are equivalent:
```
chips> D 1 CURVE data/exampleB.dat 1 2 6
chips> CURVE data/exampleB.dat X 1 Y 2 E 6
chips> CURVE data/exampleB.dat X 1 Y 2 YERR 6
```

19. ```
chips> CURVE data/exampleB.dat
chips> D 1 C 1 CURVE HISTO
```
A single curve is plotted and the curve style is changed to histogram. The following commands are equivalent:
```
chips> C 1 CURVE HISTO
chips> CURVE HISTO
chips> PLOT HISTO
chips> HISTO
```

20. ```
chips> LONGDASH
chips> WIDTH 4.0
chips> RED
chips> SYMBOL UPTRI
chips> SYMBOL BLUE
chips> SYMBOL SIZE 5.0
```
Building on the previous example, various attributes to the curve are modified. The histogram line is changed from to a wide (width 4) line of long dashes. RED changes the color of the curve line, and the three SYMBOL commands change the type, color, and size of the plot symbols.

21. ```
chips> C 1 DEL
You have no more curves in this drawing area.
chips> CURVE data/exampleB.dat 1 2 6
chips> ERRS BAR
```
All existing curves are removed, then data with symmetric Y-axis error bars is plotted. The final command modifies these error bars such that they do not have ticks on the top or bottom of the bars.

22. ```
chips> CURVE data/exampleB.dat X 1 Y 2 XUP 4 XDOWN 5 YUP 7 YDOWN 8
chips> ERRS X UP
chips> ERRS X BAR
```
Data is plotted with asymmetric x- and y-axis error bars. The command ERRS X UP causes only the positive x errorbars to be plotted. Then the errors are changed so that they do not have ticks on the top or bottom of the bars.

23. ```
chips> CURVE data/exampleA.dat
chips> CURVE DATA data/exampleA.dat YUP 5 YDOWN 6
chips> CURVE data/exampleB.dat
chips> CLEAR
```
The first CURVE command adds curve number 1 to the drawing area, from file data/exampleA.dat. The CURVE DATA command alters the curve so that it includes error bars from the same file. The third command adds curve number 2 to the drawing area, from file data/exampleB.dat. All plotting objects are then deleted, creating a new blank drawing area.

24. `chips> CURVE data/exampleB.dat X 1 Y 2 YERR 6`
    `chips> D 1 CURVE DATA data/exampleB.dat X 1 Y 2 YUP 7 YDOWN 8`

    X, y, and symmetric y errors are plotted from an ASCII data file. The curve is then altered to have separate positive and negative y errors plotted.

2.11.6 ARGUMENT LIST

The `CURVE` command (the *ChIPS* command not the *S-Lang* function) can take many arguments:

```
chips> [D #] [C #] {CURVE | PLOT} DATA <filename> \
      [X #] [Y #]} [{XERR # | XUP # | XDOWN # | XDN #} \
      [{E # | YERR # | YUP # | YDOWN # | YDN #} [SCOL #]
```

The filename can include the path to the file (either relative or absolute) and - if a FITS file - any virtual file syntax as described in "ahelp dmsyntax". If a Data Model filter is used (*i.e.* something within "[]") then the whole file name has to be enclosed in "" characters.

```
Argument: ABSOLUTE or ABS
        Description: errors are not relative to the data value,
                     they are plotted as absolute values
Default: RELATIVE
```

```
Argument: <color>
Description: Color of the axes
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE, YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <coor>
        Description: axis coordinates
Options: X, Y
Default: both X and Y
```

```
Argument: <curvestyle>
        Description: style of curve
Options: NOLINE - no connecting line between data points,
         SIMPLELINE - line connecting data points,
         STEP - stepped connection between data points (alias for HISTO)
Default: NOLINE
```

```
Argument: E #
```

```
Options: column number for the symmetric error on y
Default: integer numbers


Argument: <errorstyle>
        Description: style for errorbars
Options: STANDARD - errorbar with top and bottom ticks,
         BAR - errorbar only (no ticks on top or bottom)
Default: STANDARD


Argument: <errortick_size_value>
        Description: size of top and bottom errorbar ticks
Options: real numbers
Default: 3.0


Argument: <errortype>
        Description: type of errorbars
Options: BOTH - both positive and negative errorbars,
         NONE - no error bars,
         UP - positive errorbar only,
         {DOWN | DN} - negative errorbar only
Default: BOTH


Argument: <linestyle>
        Description: style for lines
Options: SOLID, DOT, DASH, LONGDASH, DOTDASH, DOTLONGDASH, DASHLONGDASH
Default: SOLID


Argument: {RELATIVE | REL}
Options: errors are relative to the data value
Default: RELATIVE


Argument: <scale_value>
        Description: plot scaling factor
Options: real numbers
Default: 1.0


Argument: SCOL #
Options: column number for the symbol specification
Default: integer numbers (see below for list)
```

```
Argument: <symbol_size_value>
        Description: size of symbols factor
Options: real numbers
Default: 3.0


Argument: <symbolstyle>
        Description: style for symbols
Options: BLOCK, CIRCLE, CROSS, DIAMOND, SOLIDDIAMOND, DOWNTRI,
         SOLIDDOWNTRI, UPTRI, SOLIDUPTRI, SQUARE, POINT, BIGPOINT,
         NONE
Default: CROSS


Argument: <width_value>
        Description: curve thickness
Options: real numbers
Default: 1.0


Argument: X #
Description: column number for the X axis
Options: integer numbers
Default: X 1


Argument: XERR #
Description: column number for the symmetric error on x
Options: integer numbers


Argument: XUP #
Description: column number for the positive error on x
Default: integer numbers


Argument: {XDOWN | XDN} #
Description: column number for the negative error on x
Options: integer numbers


Argument: Y #
Description: column number for the Y axis
Options: integer numbers
Default: Y 2
```

```
Argument: YERR #
Description: column number for the symmetric error on y
Options: integer numbers
```

```
Argument: YUP #
Description: column number for the positive error on y
Options: integer numbers
```

```
Argument: {YDOWN | YDN} #
Description: column number for the negative error on y
Options: integer numbers
```

2.11.7 SCOL values

The following table lists the possible values for the SCOL column.

Value	Symbol type
1 - 19	dot
20 - 29	horizontal dash
30	open triangle
31	inverted Y
32	starred triangle
33	solid triangle
40	open square
41	X
42	starred square
43	solid square
AB	The two-digit number AB means that the symbol (a polygon) has A sides and that these sides should be connected according the the value of B: 0=open, 1=skeletal, 2=starred, and 3=solid. No symbol will be produced if B is greater than 3.

2.12 D

Defines the current drawing area.

```
chips> D {# [,#, ...] | #:# | ALL}
```

where # is an integer argument specifying drawing area number, and #:# specifies an inclusive range of drawing area numbers.

Note that wherever the "D #" is used, the above extended syntax may be utilized to alter multiple drawing areas at once. If a non-existent drawing area is specified, *ChIPS* will act on the current drawing area.

Examples:

1. ```
   chips> SPLIT 2
   chips> SPLIT GAP 0.04
   chips> D 1 CURVE data/exampleA.dat
   chips> D 2 CURVE data/exampleB.dat
   chips> TWOAXESFRAME
   ```

   The drawing area is first divided into two parts and and separated by a small amount (0.04). The first curve is plotted in drawing area 1, and the second in drawing area 2. Drawing area 2 is now the current drawing area, since it is the most recently created. The command `TWOAXESFRAME` acts on the current drawing area, changing the style of the axes.

2. ```
   chips> D 2
   chips> TICKS X 10.0
   ```

   First, drawing area number is defined as the the current drawing area. The `TICKS X 10.0` command acts on drawing area number 2, changing the tick spacing to units of 10.

3. ```
   chips> D 1,2
   chips> AXES BLUE
   ```

   "D 1,2" defines drawing areas 1 and 2 as the current drawing areas, so that the following `AXES BLUE` command acts on both of them. The following commands are equivalent:

   ```
   chips> D 1:2 AXES BLUE
   chips> D ALL AXES BLUE
   ```

4. ```
   chips> SPLIT 3
   chips> SPLIT GAP 0.1
   chips> D 1 CURVE data/exampleA.dat
   chips> D 2 CURVE data/exampleB.dat
   chips> D 3 CURVE data/example.dat
   chips> D ALL LIMITS X 0 20
   ```

   Three drawing areas are created with a large separation between them. A separate curve is plotted in each area, then they are all set to have the same X-axis limits.

2.13 DELETE

Deletes plotting objects (drawing area, curves, lines and labels) from a plot.

```
chips> [D #] [C #] [LN #] [L #] DELETE
chips> DELETE [DRAWAREA] [#]
```

The first command deletes various plotting objects within a drawing area. To remove the entire drawing area and its contents, use the second command.

```
Argument: D #
Description: drawing area number designation
Options: integer number
```

See the D command (Section 2.12) for more information about this argument.

```
Argument: C #
Description: curve number designation
Options: integer number
```

See the C command (Section 2.7) for more information about this argument.

```
Argument: LN #
Description: line number designation
Options: integer number
```

```
Argument: L #
Description: label number designation
Options: integer number
```

Note that if a non-existent plotting object is specified, *ChIPS* will act on the current plotting object.

The PACK command (Section 2.31) controls how plotting objects are renumbered after deletions are made.

Examples:

1. ```
   chips> SPLIT 2
   chips> SPLIT GAP 0.05
   chips> D 1 CURVE data/exampleA.dat
   chips> D 1 CURVE data/exampleB.dat
   chips> D 1 LABEL 2.0 60.0 "Drawing Area 1"
   chips> D 1 LABEL 2.0 20.0 "Curve 1"
   chips> D 1 LABEL 5.5 10.0 "Curve 2"
   chips> D 2 CURVE data/exampleB.dat
   chips> D 2 LABEL 2.0 3.4 "Drawing Area 2"
   chips> D 1 C 2 DELETE
   Your new Current Curve number is: 1
   chips> D 1 L 3 DELETE
   Your new Current Label number is: 2
   chips> D 2 DELETE
   Your new Current Draw Area number is: 1
   ```

   The drawing area is first divided into two parts. Three curves are plotted - two in drawing area 1 and one in drawing area 2 - and labeled. The command D 1 C 2 DELETE deletes the second curve that was plotted in drawing area number 1; the next DELETE command deletes the label for that curve. The final command removes drawing area number 2 enitrely.
   ```

2. ```
   chips> CLEAR
   chips> CURVE data/exampleA.dat
   chips> LABEL 4.0 5.0 "Label 1"
   chips> LABEL 2.0 40.0 "Label 2"
   chips> LABEL 4.0 60.0 "Label 3"
   chips> LABEL 3.0 20.0 "Label 4"
   chips> L 2,4 DELETE
   Your new Current Label number is: 2
   chips> L 2:4 DELETE
   Your new Current Label number is: 1
   ```

   A single curve is plotted and four labels are created. The command L 2,4 DELETE deletes label numbers 2 and 4. By default, renumbering of plotting objects is turned on, so the remaining labels are renumbered as 1 and 2. Issuing L 2:4 DELETE deletes any labels numbered 2 through 4, *i.e.* “Label 3” is deleted.

3. ```
   chips> CURVE data/exampleA.dat
   chips> LABEL 4.0 5.0 "Label 1"
   chips> LABEL 2.0 40.0 "Label 2"
   chips> L 2 DEL
   Your new Current Label number is: 1
   chips> L 2 SIZE 4.0
   ```

   A curve is plotted and two labels are created. The L 2 DEL command removes label number 2 (*i.e.* “Label 2”). Since there is no longer a label number 2 upon which to act, the L 2 SIZE 4.0 command is slightly misleading; this command will act on label number 1, as explained in the DESCRIPTION.

2.14 DISPLAY

Causes the specified 2-D data to be displayed, using the default imaging display tool (*i.e.* ds9)

```
chips> DISPLAY <filename>
```

The <filename> is the name of an image data file, and may include a directory path. Currently, only 2-D FITS images are supported.

Examples:

1. ```
   chips> DISPLAY data/example_img.fits
   ```

   Displays the image contained in data/example_img.fits.

2.15 DRAWAREA

Specifies the location for a new drawing area.
   ```

```
chips> DRAWAREA <x1> <x2> <y1> <y2>

Argument: <x1>
Description: X coordinate of lower left corner
Options: real numbers between 0.0 and 1.0

Argument: <x2>
Description: X coordinate of upper right corner
Options: real numbers between 0.0 and 1.0

Argument: <y1>
Description: Y coordinate of lower left corner
Options: real numbers between 0.0 and 1.0

Argument: <y2>
Description: Y coordinate of upper right corner
Options: real numbers between 0.0 and 1.0
```

Examples:

1. `chips> DRAWAREA 0.25 0.9 0.5 0.9`
   `chips> D 1 CURVE data/example.dat`
   The drawing area location is set and then a curve is plotted.

2.16 ERRS

Specifies the appearance of errorbars.

```
chips> [D #] [C #] ERRS [<coor>] <color>
chips> [D #] [C #] ERRS [<coor>] <errorstyle>
chips> [D #] [C #] ERRS [<coor>] <errortype>
chips> [D #] [C #] ERRS [<coor>] <linestyle>
chips> [D #] [C #] ERRS [<coor>] {ABSOLUTE | RELATIVE}
chips> [D #] [C #] ERRS [<coor>] SIZE <errortick_size_value>

Argument: {ABSOLUTE | ABS}
Description: errors are not relative to the data value; instead
             they are plotted as absolute values
Default: RELATIVE

Argument: C #
Description: curve number designation
Options: integer numbers
Default: current curve
```

See the `C` command (Section 2.7) for more information about this argument.

```
Argument: <color>
Description: axes color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED,  WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: both X and Y
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <errorstyle>
Description: sytle for errorbars
Options: STANDARD - errorbar with top and bottom ticks
         BAR - errorbar only, no ticks on top or bottom
Default: STANDARD
```

```
Argument: <errortype>
Description: type of errorbars
Options: BOTH - both positive and negative errorbars
         NONE - no error bars
         UP - positive errorbar only
         {DOWN | DN} - negative errorbar only
Default: BOTH
```

```
Argument: <errortick_size_value>
Description: size of top and bottom errorbar ticks
Options: real numbers
Default: 3.0
```

```
Argument: <linestyle>
Description: style for lines
Options: SOLID, DOT, DASH, LONGDASH, DOTDASH, DOTLONGDASH,
         DASHLONGDASH
Default: SOLID
```

```
Argument: {RELATIVE | REL}
Description: errors are relative to the data value
Default: RELATIVE
```

The command:

```
chips> [D #] [C #] ERRS [<coor>] {ABSOLUTE | RELATIVE}
```

affects all subsequent plots, not just the existing plot.

Figure 2.2 is an illustration of the different <`errortype`> options. Figure 2.3 is an illustration of the different <`errorstyle`> options.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "`ahelp chips`" for further details.

Examples:

1. ```
   chips> SPLIT 2
   chips> D 1 CURVE data/exampleA.dat YUP 5 YDOWN 6
   chips> D 1 CURVE data/exampleB.dat YUP 7 YDOWN 8
   chips> D 2 CURVE data/exampleA.dat XUP 3 XDOWN 3
   chips> D 2 CURVE data/exampleB.dat XUP 4 XDOWN 5
   chips> D 2 C 1 ERRS X UP
   chips> D 2 C 1 ERRS BAR
   ```

   Two curves with Y error bars are plotted in drawing area number 1, and two curves are plotted in drawing area number 2 with X error bars. The error bars for curve 1 of drawing area 2 are then changed to be positive error bars only, having no tick marks on the tops or bottoms of the bars.

2. ```
   chips> D 1 C 1 ERRS Y UP
   chips> D 1 C 1 ERRS BAR
   ```

   Building on the previous example, the Y error bars plotted in drawing area 1 for curve 1 are modified.

3. ```
   chips> ERRS ABS
   chips> D 1 CURVE data/exampleA.dat YUP 5 YDOWN 6
   chips> ERRS REL
   chips> D 1 CURVE data/exampleAa.dat YUP 5 YDOWN 6
   ```

   The first `CURVE` command (Section 2.11) is preceded by the ERRS ABS command, which will set all errors to be plotted as absolute (instead of relative) to the data points. The second CURVE command is preceded by the ERRS REL command, which will set all errors to plotted as relative to the data.

2.17 EXIT — QUIT

Terminates the program. The command QUIT is equivalent.
   ```

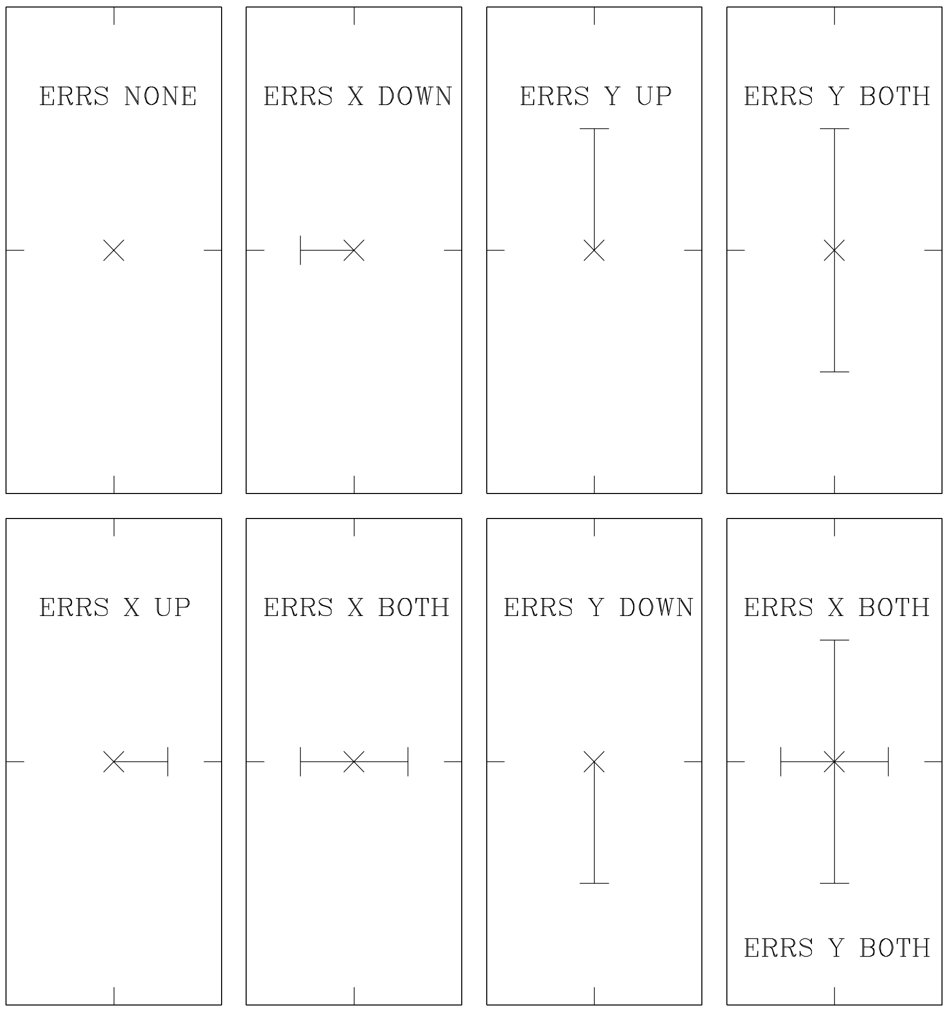

Figure 2.2: <**errortype**> Options

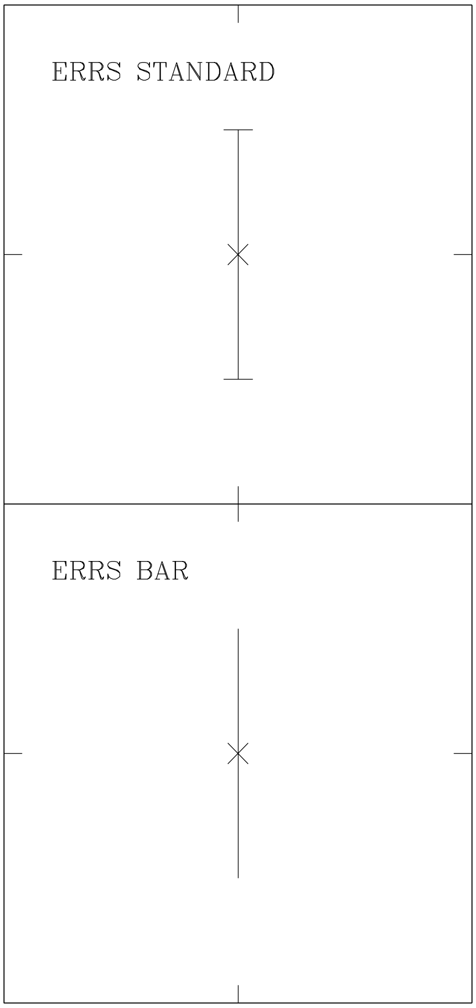

Figure 2.3: <**errorstyle**> Options

```
chips> EXIT
chips> QUIT
```

Examples:

1. `chips> EXIT`
   `unix%`

   ChIPS has been closed, returning the user to the unix prompt.

2.18 FONT

Changes the default global font.

```
chips> FONT <font>
```

```
Argument: <font>
Description: font type designation
Options: {GREEK | GR} (Greek), {ITALIC | IT} (Italic),
         {OLDENGLISH | OE} (Old English), {ROMAN | RM} (Roman),
         {TINY | TI} (Tiny)
Default: ROMAN
```

This command affects all drawing areas.

The <text> argument of the commands **LABEL** (Section 2.22), **XLABEL** (Section 2.54), **YLABEL** (Section 2.55), **ZLABEL** (Section 2.56), and **TITLE** (Section 2.49) may contain TeX syntax for formatting text and creating mathematical expressions. This TeX syntax may include mixed font settings, as shown in one of the EXAMPLES.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> XLABEL "X-axis"
   chips> YLABEL "Y-axis"
   chips> TITLE "Title"
   chips> LABEL 2.0 2.5 "Plot label"
   chips> XLABEL SIZE 2.0
   chips> YLABEL SIZE 2.0
   chips> TITLE SIZE 2.0
   chips> LABEL SIZE 2.0
   chips> FONT GR
   ```
   ```

```
chips> FONT RM
```

After creating a plot and adding labels and a title, the global font is changed first to Greek. The final command changes the global font back to the default Roman. The following commands are equivalent when changing fonts:

```
chips> FONT GREEK
chips> FONT ROMAN
```

2. `chips> D 1 LABEL 2.0 50.0 "\oe This \rm is \gr Greek"`

   This LABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where “This” is in Old English font, “is” is in Roman font, and “Greek” is in the Greek font.

2.19 GRIDS

Adds grid lines to a drawing area and/or changes grid attributes.

```
chips> [D #] GRIDS [<gridtype>] <color>
chips> [D #] GRIDS [<gridtype>] [<coor>] <boolean>
chips> [D #] GRIDS [<gridtype>] <linestyle>
```

```
Argument: <boolean>
Description: drawing area grids on or off
Options: {ON | OFF}
Default: ON
```

```
Argument: <color>
Description: axes color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE, YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: both X and Y
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <gridtype>
Description: type of grid markings
Options: MAJ - grid lines along major tick marks
         MIN - grid lines along minor tick marks
Default: both MAJ and MIN
```

```
Argument: <linestyle>
Description: style for lines
Options: SOLID, DOT, DASH, LONGDASH, DOTDASH, DOTLONGDASH,
         DASHLONGDASH
Default: SOLID
```

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> CURVE data/example.dat`
   `chips> GRIDS ON`

   A curve is plotted, then the grid lines are turned on.

2. `chips> GRIDS MAJ Y ON`
   `chips> GRIDS MIN X ON`

   Grid lines along the major Y-axis tick marks are turned on, while the lines along the minor X-axis tick marks are turned on.

3. `chips> GRIDS MAJ BLUE`
   `chips> GRIDS MIN CYAN`
   `chips> GRIDS LONGDASH`

   The major tick marks are changed to the color blue, and the minor tick marks are changed to the color cyan. Finally, all grid line styles are changed from a solid line to a long dash.

2.20 INFO

Prints summary information about the current plotting objects.

```
chips> INFO
```

Examples:

1. `chips> CURVE data/example.dat`
   `chips> GRIDS ON`
   `chips> AXES YELLOW`
   `chips> INFO`

```
Drawing Area  #1  << CURRENT DRAWING AREA
(Location: 0.15 0.9 0.1 0.9)
(Limits  : -0.45 20.45 0.915761 3.99156)
(Axes    :  fouraxes   color: yellow  width: 1)
        Curve #1  /data/example.dat
                 SYMBOL: defaultcolor  cross
                 CONNECTOR: noline << CURRENT CURVE
```

A plot is created and some of the attributes are modified. The plot summary is then printed with the INFO command.

2.21 L

Defines the current label.

```
chips> [D #] L {# [,#, ...] | #:# | ALL}
```

where # is an integer argument and #:# specifies an inclusive range of label numbers.

See the `D` command (Section 2.12) for more information about this argument.

If a non-existent label is specified, *ChIPS* will act on the current label.

Examples:

1. ```
   chips> CURVE data/exampleA.dat
   chips> CURVE data/exampleB.dat
   chips> LABEL 5.0 40.0 "Curve 1"
   chips> LABEL 5.0 5.0 "Curve 2"
   chips> LABEL YELLOW
   ```

   Two curves are plotted and labeled; The first label ("Curve 1") is label number 1 and the second label is number 2. Label number 2 is also the current label, since it was most recently created. The command `LABEL YELLOW` acts on the current label (2), changing the color to yellow.

2. ```
   chips> D 1 L 1
   chips> LABEL BLUE
   ```

   The command D 1 L 1 defines label number 1 as the current label, so that the command `LABEL BLUE` changes the color.

3. ```
   chips> L 2
   chips> LABEL SIZE 2.4
   ```

   Label 2 is defined as the current label, then the size is changed to 2.4.
   ```

2.22 LABEL

Adds a label to a drawing area and/or changes label attributes.

```
chips> [D #] LABEL <xcoor> <ycoor> "<text>"
```

To change label attributes:

```
chips> [D #] {L # | LABEL} "<text>"
chips> [D #] L # <xcoor> <ycoor>
chips> [D #] {L # | LABEL} ANGLE <rot_angle>
chips> [D #] {L # | LABEL} <color>
chips> [D #] {L # | LABEL} SIZE <size_value>
```

The LABEL command can take many arguments:

```
Argument: <color>
Description: label color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE, YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: L #
Description: label number designation
Options: integer numbers
Default: current label
```

```
Argument: <rot_angle>
Description: rotation angle in degrees
Options: real numbers
Default: 0.0
```

```
Argument: <size_value>
Description: label size
Options: real numbers
Default: 1.0
```

```
Argument: <text>
Description: the text for the label, which must be in quotes
             (" ").  The <text> may contain TeX syntax
             for formatting text, creating mathematical expressions,
             and mixing font settings.
Options: quoted character string


Argument: <xcoor>
Description: X coordinate of start location for label
Options: real numbers


Argument: <ycoor>
Description: Y coordinate of start location for labels
Options: real numbers
```

If <xcoor> and <ycoor> are not included, the current label is replaced. These arguments *must* be included when adding the first label.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. ```
   chips> CURVE data/exampleB.dat
   chips> D 1 LABEL 5.0 1.2 "Label"
   ```
   After a curve has been plotted, a label is added at position (5.0, 1.2).

2. ```
   chips> D 1 LABEL "NewLabel"
   ```
   The text of the label created in the previous example is replaced with "NewLabel".

3. ```
   chips> D 1 LABEL 10.0 3.0 "x = \frac{y+z/2}{y^{2}+3}"
   ```
   A mathematical label is added at position (10.0, 3.0) by using TeX syntax.

4. ```
   chips> D 1 LABEL 10.0 1.5 "\oe This \rm is \gr Greek"
   ```
   This LABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where "This" is in Old English font, "is" is in Roman font, and "Greek" is in the Greek font.

5. ```
   chips> CLEAR
   chips> CURVE data/exampleB.dat
   chips> D 1 LABEL 5.0 1.2 "Label"
   chips> L 1
   chips> LABEL ANGLE 45.0
   chips> L 1 10.0 1.2
   chips> LABEL SIZE 2.0
   ```
   A curve is plotted and labeled. The angle of the label is changed to 45 degrees, then it is moved and increased in size.
   ```

6. `chips> LABEL 5.0 3.0 "Label 3"`
   `chips> LABEL 5.0 1.5 "Label 4"`
   `chips> L 2,3 ANGLE 45.0`
   `chips> L 1:3 SIZE 1.5`

   Two more labels are added to the plot from the previous example, then the command L 2,3 ANGLE 45.0 changes the angle of both of them to 45 degrees. The final command changes the size of labels numbered 1 through 3 (all labels in this case) to 1.5.

2.23 LEVELS

Sets the contour levels in a contour plot.

```
chips> [D #] [C #] LEVELS <lev1> <lev2> ... <levN>
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: C #
Description: curve number designation
Options: integer number
Default: current curve
```

See the `C` command (Section 2.7) for more information about this argument.

```
Argument: <lev1> <lev2> ... <levN>
Description: numbers specifying Z axis values of each contour level
Options: real numbers
```

The <lev1> value specifies the Z-axis data value of the first contour line, <lev2> specifies the value of the second contour line, and so on. See the `CONTOUR` command (Section 2.10) for information on displaying contour plots.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**`ahelp chips`**" for further details.

Examples:

1. `chips> CONTOUR data/example3D.sorted.dat 2 4 20`

```
chips> LEVELS 3 5 20 100
```

The ASCII data file data/example3D.sorted.dat is plotted as a contour plot. The first contour line is drawn along Z-axis data values of 2, the second along values of 4, and the third along values of 20. The LEVELS command changes the contour levels to 3, 5, 20, and 100 for contour lines 1 thru 4, respectively.

2.24 LIMITS

Sets the ranges to be displayed in the axes.

```
chips> [D #] LIMITS [<coor>] {<min> | AUTO | *} {<max> | AUTO | *}
chips> [D #] LIMITS {<xmin> | AUTO | *} {<xmax> | AUTO | *} \
{<ymin> | AUTO | *} {<ymax> | AUTO | *} \
[{<zmin> | AUTO | *} {<zmax> | AUTO | *}]
```

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: X
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the D command (Section 2.12) for more information about this argument.

```
Argument: <min>
Description: lower limit for axis range
Options: real numbers
Default: AUTO
```

```
Argument: <max>
Description: upper limit for axis range
Options: real numbers
Default: AUTO
```

```
Argument: <xmin>
Description: lower limit for X-axis range
Options: real numbers
Default: AUTO
```

```
Argument: <xmax>
Description: upper limit for X-axis range
Options: real numbers
Default: AUTO

Argument: <ymin>
Description: lower limit for Y-axis range
Options: real numbers
Default: AUTO

Argument: <ymax>
Description: upper limit for Y-axis range
Options: real numbers
Default: AUTO

Argument: <zmin>
Description: lower limit for Z-axis range
Options: real numbers
Default: AUTO

Argument: <zmax>
Description: upper limit for Z-axis range
Options: real numbers
Default: AUTO
```

The limit specifications come in pairs, with the first being the lower limit. A limit specification may be a number, an asterisk (*), or the argument AUTO. A '*' indicates no change in the current limit, while AUTO allows the underlying plotting package to select the limit value.

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

The limits of the x and y axes in a contour plot cannot be modified in the current version of *ChIPS*.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "`ahelp chips`" for further details.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> LIMITS X 5.0 15.0
   chips> LIMITS Y 2.0 4.0
   chips> LIMITS X AUTO AUTO
   chips> LIMITS Y AUTO AUTO
   ```
   The first two LIMITS commands change the displayed ranges of the X and Y axes, respectively. The final two commands instruct the plotting package to determine the X and Y limits. The two pairs of commands (4 lines) may be reduced to two lines:
   ```

```
chips> LIMITS 5.0 15.0 2.0 4.0
chips> LIMITS AUTO AUTO AUTO AUTO
```

2. `chips> LIMITS * 15.0 * 4.5`

   This command preserves the minimum value of each axis, but changes the maximum of the X-axis to 15.0 and of the Y-axis to 4.5.

2.25 LINE

Adds a line to a drawing area and/or changes line attributes.

```
chips> [D #] LINE <x1> <y1> <x2> <y2>
```

To change line attributes:

```
chips> [D #] LN # <x1> <y1> <x2> <y2>
chips> [D #] {LN # | LINE} <color>
chips> [D #] {LN # | LINE} <linestyle>
chips> [D #] {LN # | LINE} WIDTH <width_value>
```

The LINE command can take many arguments:

```
Argument: <color>
Description: line color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE, YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <linestyle>
Description: style for line
Options: SOLID, DOT, DASH, LONGDASH, DOTDASH, DOTLONGDASH, DASHLONGDASH
Default: SOLID
```

```
Argument: LN #
Description: line number designation
Options: integer numbers
```

```
Argument: <width_value>
Description: line thickness
Options: real numbers
Default: 1.0


Argument: <x1>
Description: X coordinate of start location for line
Options: real numbers


Argument: <x2>
Description: X coordinate of end location for line
Options: real numbers


Argument: <y1>
Description: Y coordinate of start location for line
Options: real numbers


Argument: <y2>
Description: Y coordinate of end location for line
Options: real numbers
```

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> CURVE data/exampleB.dat`
   `chips> D 1 LINE 1.0 1.8 3.0 1.8`

   After a curve has been plotted, a line is added that begins at (5.0, 1.0) and ends at (10.0, 2.0).

2. `chips> CURVE data/exampleB.dat`
   `chips> D 1 LINE 1.0 1.8 3.0 1.8`
   `chips> LINE BLUE`
   `chips> LN 1 1.0 1.8 10.0 1.8`
   `chips> LINE DOTDASH`
   `chips> LINE WIDTH 5.0`

   A curve and line are plotted, and the color of the line is changed to blue. Then the LN command changes the endpoint of the line. The final two commands switch the line to a dot-dash style and increase the width to 5.0.

2.26 LINEAR

Changes one or more axes to linear scale.

```
chips> [D #] LINEAR [<coor>]
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: both X and Y
```

The command `LOG` (Section 2.30) is used to return an axis to log scale.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "`ahelp chips`" for further details.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> LOG X
   chips> LOG
   chips> LINEAR
   ```

   The data is plotted, then the X-axis is changed to logarithmic scale. The second LOG command changes both the X and Y axes to logarithmic scale, and the LINEAR command returns both axes to linear scale.

2.27 LIST

Lists the contents of datafiles using the CIAO tool dmlist.

```
chips> LIST <filename> [<dm_list_parameter_specifications> [| <arg>]]
```

In addition to the <filename>, the input may specify the dmlist option parameter (<dm_list_parameter_specifications>) and any pipe-supporting Unix command (<arg>, *e.g.* less).

For more information, see "`ahelp dmlist`".

Examples:
```

1. `chips> acisf01838N001_evt2.fits opt=blocks`

```
--------------------------------------------------------------------------------
Dataset: acisf01838N001_evt2.fits
--------------------------------------------------------------------------------

     Block Name                             Type         Dimensions
--------------------------------------------------------------------------------
Block    1: PRIMARY                           Null
Block    2: EVENTS                            Table        14 cols x 78208    rows
Block    3: GTI7                              Table         2 cols x 1        rows
Block    4: GTI0                              Table         2 cols x 1        rows
Block    5: GTI1                              Table         2 cols x 2        rows
Block    6: GTI2                              Table         2 cols x 1        rows
Block    7: GTI3                              Table         2 cols x 1        rows
Block    8: GTI6                              Table         2 cols x 1        rows
```

The blocks of the file acisf01838N001_evt2.fits are returned.

2. `chips> LIST primary/acisf01838N001_evt2.fits cols | less`

```
--------------------------------------------------------------------------------
Columns for Table Block EVENTS
--------------------------------------------------------------------------------

ColNo  Name                Unit        Type             Range
   1   time                s            Real8          84244214.7546979934: 842
   2   ccd_id                           Int2           0:9                   CCD
   3   node_id                          Int2           0:3                   CCD
   4   expno                            Int4           0:2147483647          Exp
   5   chip(chipx,chipy)   pixel        Int2           1:1024                Chi
   6   tdet(tdetx,tdety)   pixel        Int2           1:8192                ACI
(etc.)
```

The columns of the data file, which includes a pathname, are piped to “less” and printed to the screen.

2.28 LN

Defines the current line.

```
chips> [D #] LN {# [,#, ...] | #:# | ALL}
```

where # is an integer argument and #:# specifies an inclusive range of line numbers.

See the `D` command (Section 2.12) for more information about this argument.

Note that if a non-existent line is specified, *ChIPS* will act on the current line.

Examples:

1. `chips> CURVE data/example.dat X 1 Y 2`
   `chips> LINE 2.0 1.0 2.0 4.0`
   `chips> LINE 1.25 5.0 1.8 5.0`
   `chips> LINE DASH`

   A curve is plotted and two lines are placed on the plot. The first line is designated line number 1 and the second is line number 2; Line 2 is also the current line, since it is the most recently created line. The command `LINE DASH` acts on the current line, changing it to a dashed line.

2. `chips> D 1 LN 1`
   `chips> LINE DASH`

   The command D 1 LN 1 defines line number 1 as the current line, then the `LINE DASH` command changes it to a dashed line.

3. `chips> LN 2`
   `chips> LINE WIDTH 4.0`

   Line 2 is defined as the current line. The `LINE WIDTH 4.0` command acts on the line, changing it to a width of 4.0.

2.29 LOCATION

Specifies the location for an existing drawing area.

```
chips> LOCATION <x1> <x2> <y1> <y2>
chips> [D #] LOCATION <x1> <x2> <y1> <y2>
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <x1>
Description: X coordinate of lower left corner
Options: real numbers between 0.0 and 1.0
Default: 0.15
```

```
Argument: <x2>
Description: X coordinate of upper right corner
Options: real numbers between 0.0 and 1.0
Default: 0.9
```

```
Argument: <y1>
```

```
Description: Y coordinate of lower left corner
Options: real numbers between 0.0 and 1.0
Default: 0.1
```

```
Argument: <y2>
Description: Y coordinate of upper right corner
Options: real numbers between 0.0 and 1.0
Default: 0.9
```

The origin - (0,0) - is at the lower left corner.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> D 1 CURVE data/example.dat`
   `chips> LOCATION 0.25 0.9 0.5 0.9`
   A curve is plotted, then the drawing area location is modified.

2.30 LOG

Changes one or more axes to logarithmic scale.

```
chips> [D #] LOG [<coor>]
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <coor>
Description: axis coordinates
Options: X, Y
Default: both X and Y
```

Note that a warning is returned when a LOG command is issued for data containing negative or zero values: "`Warning:  negative and zero values ignored in log scale`".

The command `LINEAR` (Section 2.26) is used to return an axis to linear scale.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1.
```
chips> CURVE data/example.dat
chips> LOG X
chips> LOG
chips> LINEAR
```
   The data is plotted, then the X-axis is changed to logarithmic scale. The second LOG command changes both the X and Y axes to logarithmic scale, and the LINEAR command returns both axes to linear scale.

2.31 PACK

Renumbers plotting objects after a delete operation.

```
chips> PACK
chips> PACK {ON | OFF}
chips> PACK DRAWAREA
chips> [D #] PACK {CURVE | LABEL | LINE}
```

See the `D` command (Section 2.12) for more information about this argument.

Plotting objects are assigned consecutive number designations, in the order in which they are created. By default, renumbering of plotting objects is turned on. This means that the objects are renumbered whenever one is removed.

Issuing this command with no arguments forces a renumbering of all plotting objects. If an argument is given, however, the specified plotting objects (drawing areas, curves, labels, and lines) are renumbered independently.

Examples:

1.
```
chips> SPLIT 2
chips> D 1 CURVE data/exampleA.dat
chips> D 2 CURVE data/exampleB.dat
chips> D 1 DEL
Your new Current Draw Area number is: 1
chips> D 1 TWOAXES
```
   The drawing area is `split in two` and a curve is plotted in each area. When drawing area number 1 is removed, drawing area number 2 becomes the current drawing area and is renumbered to be 1 (since renumbering is on by default). The command `D 1 TWOAXES` acts on the remaining drawing area.

2.
```
chips> SPLIT 2
```

```
chips> D 1 CURVE data/exampleA.dat
chips> D 2 CURVE data/exampleB.dat
chips> PACK OFF
chips> D 1 DEL
Your new Current Draw Area number is: 2
```

Again, two curves are plotted, each in its own drawing area. The PACK OFF command causes plotting objects not to be renumbered, so drawing area number 2 becomes the current drawing area without changing number.

3.
```
chips> D 1 CURVE data/exampleB.dat
chips> LABEL 5.0 1.5 "Label 1"
chips> LABEL 10.0 2.0 "Label 2"
chips> LABEL 15.0 3.0 "Label 3"
chips> LABEL 5.0 3.6 "Label 4"
chips> PACK OFF
chips> L 2 DEL
Your new Current Label number is: 3
chips> L 3 DEL
Your new Current Label number is: 4
chips> PACK LABEL
chips> L 1 ANGLE 45.0
chips> L 2 SIZE 2.0
```

A curve is plotted and four labels are added to the drawing area. The two `DEL commands` remove labels 2 and 3; since the command PACK OFF was issued, the remaining labels are not renumbered. The PACK LABELS command causes labels 1 and 4 to be renumbered to 1 and 2, which are then modified by the label attribute commands.

2.32 PAGESIZE

Sets the page dimensions for hardcopy plots.

```
chips> pagesize <width> <length> <unit>
```

In this context, "hardcopy" refers to both sending the plot directly to a printer and printing it to a postscript file.

```
Argument: <width>
Description: width of the hardcopy plot
Options: real numbers
Default: 8.0
```

```
Argument: <length>
Description: length of the hardcopy plot
Options:  real numbers
Default: 8.0
```

```
Argument: <unit>
Description: unit for <width> and <length>
Options: cm, inch, mm
Default: inch
```

The postscript drivers work well for plots no bigger than 8x8 inch or portrait plots up to about 8x10 inches. Any other setup causes problems due to inconsistant handling of margins that causes errors in calculating the bounding box. The result is that the output of "print postfile" may be clipped even if the pagesize is set correctly; this is most likely to affect plots oriented as "landscape" layout.

If creating a smaller plot is not an option, try one of these workarounds:

- edit the bounding box by hand in PS file
- delete the bounding box definition from the PS file
- gv has the option to change or ignore the bounding box when viewing a PS file

More information on creating hardcopies is available from "**ahelp chips print**".

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Unlike most other *ChIPS* commands, the "pagesize" command modifies the state object. This means that once the desired size has been set, the command does not need to be reissued. The current page size may also be determined by checking the state object, as shown in the EXAMPLES section.

Examples:

1. ```
   chips> rx = [1:10:0.1]
   chips> ry = sin(rx)
   chips> curve x rx y ry
   chips> pagesize 4.0 4.0
   chips> print postfile half.ps
   ```

   First, a simple plot is created. The page size is set to 4x4 with the default unit of inches (note that it is not necessary to include the unit if it is unchanged). Finally, a postscript file is created.

2. ```
   chips> print(chips)
   ...
   pagewidth        =  4
   pagelen          =  4
   unit             =  inch
   ...
   chips> pagesize 15 15 cm
   chips> print(chips)
   ...
   pagewidth        =  15
   pagelen          =  15
   unit             =  cm
   ```

...

Building on the previous example, the height and width of the plot are modified, with the new dimensions given in cm. This also illustrates that changing the page size on the command line updates the *ChIPS* state object fields.

2.33 PICKPOINTS

Starts the interactive cursor mode, which reads out the cursor position.

```
chips> [D #] PICKPOINTS
chips> [D #] PICKPOINTS <value>
```

where <value> is an integer number of points to pick (defaults to 1 if not given).

See the `D` command (Section 2.12) for more information about this argument.

The PICKPOINTS command puts the user into interactive cursor mode. Points may be selected by clicking with the left mouse button or using the spacebar; the coordinates of selected points will be printed to the screen. If the number of points to be selected is not specified with the <value> argument, then interactive mode will continue until the character q is entered or a right mouse click is given.

Examples:

1. ```
   chips> CURVE example.dat
   chips> PICKPOINTS 3

   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (2.336950, 1.447520)
   Point picked: (3.479965, 1.770423)
   Point picked: (4.568552, 2.055780)
   chips>
   ```
   In this example, the PICKPOINTS 3 command allows the user to select three points. After the third click, the user is returned to the *ChIPS* command line.

2. ```
   chips> PICKPOINTS
   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (1.466081, 1.184691)
   Point picked: (2.554667, 1.462538)
   Point picked: (3.534395, 1.770423)
   Point picked: (4.568552, 2.055780)
   Point picked: (5.602708, 2.386193)
   chips>
   ```
   In this example, the PICKPOINTS command allows the user to select an indefinite number of points.

2.34 PLOT — CURVE

Plots a curve in a drawing area, or changes curve attributes.

See the `CURVE` command (Section 2.11), for further information and examples.

2.35 PRINT

Creates a hardcopy of the current plot.

```
chips> PRINT {[<device>] | POSTFILE <filename>}
```

If no printer device is specified, the output will be sent to the default printer as specified in the user's environment. The "POSTFILE <filename>" argument creates a PostScript file in the current directory if no pathname is included.

There are two new commands in CIAO 3.0 that may be used to configure the look of the output: `COLORSYS` (Section 2.9) sets the color system (RGB, CMYK, grayscale) and `PAGESIZE` (Section 2.32) indicates the page dimensions of the hardcopy plot.

Note that the new `COLORSYS` command deprecates the former method of creating CMYK plots, namely:

```
chips> PRINT "postfileCMYK <filename>"
```

The postscript drivers work well for plots no bigger than 8x8 inch or portrait plots up to about 8x10 inches. Any other setup causes problems due to inconsistant handling of margins that causes errors in calculating the bounding box. The result is that the output of "print postfile" may be clipped even if the pagesize is set correctly; this is most likely to affect plots oriented as "landscape" layout.

If creating a smaller plot is not an option, try one of these workarounds:

- edit the bounding box by hand in PS file
- delete the bounding box definition from the PS file
- gv has the option to change or ignore the bounding box when viewing a PS file

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "`ahelp chips`" for further details.

Examples:

1. `chips> CURVE data/example.dat`

```
chips> print(chips.colorsys)
rgb
chips> PRINT POSTFILE plot_rgb.ps
chips> COLORSYS CMYK
chips> PRINT POSTFILE plot_cmyk.ps
```

A curve is plotted and written out first as an RGB postscript file, then as a CMYK postscript file; see "`ahelp colorsys`" for an explanation of the syntax used.

2. ```
   chips> CURVE example2.dat
   chips> PRINT
   ```

   A curve is plotted and a hardcopy is printed on the system's printer.

2.36 PRISM — BROWSE

Launches *Prism*.

See the `BROWSE` command (Section 2.6), for further information and examples.

2.37 QUIT — EXIT

Terminates the *ChIPS* program.

See the `EXIT` command (Section 2.17), for an example.

2.38 REDO

Redoes the most recently undone plotting command.

```
chips> REDO [#]
```

where # is an integer number of commands to be redone.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> LIMITS X 5.0 15.0
   chips> UNDO
   chips> REDO
   ```

   A curve is plotted and the limits of the X-axis are changed. The `UNDO` command (Section 2.51) restores the limit to their original defaults. The REDO command then redoes the most recently undone command, thus changing the X-axis limits once again.

2. ```
   chips> CURVE data/example.dat
   chips> LIMITS X 5.0 15.0
   chips> LIMITS Y 10.0 15.0
   chips> AXES BLUE
   chips> LIMITS Y 10.0 25.0
   chips> UNDO 3
   chips> REDO 1
   ```

   A curve is plotted and the limits of both axes are changed. They are colored blue, then the Y-axis limits are changed again. The `UNDO` command (Section 2.51) backtracks through the previous three commands (in the order they are undone: LIMITS Y 10.0 25.0, AXES BLUE, LIMITS Y 10.0 15.0). The REDO command then redoes the most recently undone command (LIMITS Y 10.0 15.0), leaving the final plot with white axes and limits of x=5:15, y=10:15.

2.39 REDRAW

Redraws the plot or sets the redraw mode

```
chips> REDRAW
chips> REDRAW ON
chips> REDRAW OFF
```

When *ChIPS* is started the redraw mode is set to "on". This means that any command that changes the plot display - such as adding or deleting a curve or changing the plot title - will have an immediate effect. There are times when this behaviour is undesireable - such as plots of large datasets or when creating a plot from a script - and so it can be turned off by calling `BATCH OFF`. It can be restored by `BATCH ON`.

After the redraw mode is turned off, *ChIPS* commands will appear to have no affect on the current plot. The `REDRAW` command on its own will force the plot to be displayed with the current contents, or the *ChIPS* `PRINT` command can be used to create a postscript version of the current plot.

The default behaviour for *Sherpa* is to turn automatic redrawing off. This includes loading the *Sherpa* module into a *S-Lang* script.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> REDRAW OFF
   chips> SYMBOL UPTRI
   chips> SYMBOL BLUE
   chips> SYMBOL SIZE 5.0
   chips> REDRAW
   ```

   After the curve is plotted, automatic redrawing is turned off. Therefore, the effect of commands that follow are not displayed until the REDRAW command refreshes the plot.

2. ```
   chips> REDRAW ON
   chips> SIMPLELINE
   ```
   ```

Building on the previous example, automatic redrawing is restored. When the `SIMPLELINE command` is issued, the plot is redrawn with the new curve linestyle.

2.40 RELATIVESIZE — RELSIZE

Alters the relative size of drawing areas. The command RELSIZE is equivalent.

```
chips> [D #] RELATIVESIZE [<column>] <size>
```

By default, <size> is 1.0.

See the `D` command (Section 2.12) for more information about this argument.

The RELATIVESIZE command with no <column> argument will alter the relative sizes of drawing areas created with the `SPLIT` command (Section 2.43). The command changes the size of the drawing area in relation to the other drawing areas.

The RELATIVESIZE command with a <column> argument will alter the width of a column of drawing areas.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. ```
   chips> SPLIT 2
   chips> D 1 CURVE data/exampleA.dat
   chips> D 2 CURVE data/exampleB.dat
   chips> RELATIVESIZE 2.1
   ```

   The curves are first plotted in separate drawing areas. The RELATIVESIZE 2.1 command changes the relative size of the current drawing area (number 2), to be 2.1 times the size of drawing area number 1.

2. ```
   chips> RELATIVESIZE 1.0
   chips> D 1 RELATIVESIZE 3.2
   ```

   The relative sizes of the drawing areas from the previous example are returned to the default value of 1.0 (equal sizes). Drawing area 1 is then made 3.2 times the size of drawing area 2.

3. ```
   chips> clear
   chips> split 2 4
   chips> RELSIZE 1 0.5
   ```

   The current plot is cleared, then split into two columns of four drawing areas. The RELSIZE command makes the width of column one 50% of the width of column two.
   ```

2.41 RESTORE

Restores a plot with all attributes.

```
chips> RESTORE <filename>
```

The <filename> is a file previously created with the *ChIPS* `STORE` command (Section 2.44) and may contain a directory path.

Examples:

1. `chips> RESTORE myplot.chp`
   The plot saved in the myplot.chp is restored. See the `STORE` command (Section 2.44) for an example of how the plot was saved.

2.42 SKIP

Skips the designated number of lines from subsequent input files.

```
chips> SKIP <skip_value>
```

All files plotted after this command is issued will have the specified number of lines skipped.

Examples:

1. `chips> SKIP 10`
   `chips> CURVE data/exampleB.dat`
   The command SKIP 10 causes the first ten lines of the data file data/exampleB.dat to be skipped when it is read in and plotted.

2.43 SPLIT

Creates multiple drawing areas and/or arranges their locations.

```
chips> SPLIT <n>
chips> SPLIT <nx> <ny>
chips> SPLIT COLS <nc1> <nc2> ... <ncN>
```

where <n>, <nx>, <ny>, <ncN> must be integers greater than zero.

The commands create plots as follows:

- SPLIT <n> - a single column containing <n> rows of drawing areas.
- SPLIT <nx> <ny> - <nx> columns, each with <ny> rows of drawing areas. The command SPLIT 1 <ny> is equivalent to SPLIT <n>.
- SPLIT COLS <nc1> <nc2> ... <ncN> - a column for each item in the list, with each value indicating how many rows to place in that column.

Drawing areas are numbered sequentially from the upper left corner, top to bottom and left to right.

To arrange drawing area locations:

```
chips> SPLIT MAJ <coor>
chips> SPLIT GAP [<coor>] <separation>
chips> SPLIT GAP <nc> <separation>
chips> SPLIT LOCATION <x1> <x2> <y1> <y2>
```

where:

```
Argument: <coor>
Description: axis on which the split or gap is created
Options: X, Y
Default: both X and Y
```

```
Argument: <nc>
Description: column or row number, depending on the current SPLIT MAJ <coor> setting
Options: integer numbers
```

```
Argument: <separation>
Description: gap spacing
Options: real numbers
Default: 0
```

SPLIT MAJ <coor> controls whether rows or columns are used when arranging drawing areas. The default setting is SPLIT MAJ X. When SPLIT MAJ Y has been set, the basic SPLIT commands operate using rows instead of columns. Note that this command affects the currently existing plot, as shown in the examples.

SPLIT GAP [<coor>] <separation> changes the gap spacing between drawing areas to the amount specified. If no <coor> is given, both row (Y axis) and column (X axis) gaps are changed. The coordinate system runs from 0 to 1, so a <separation> <= 0.1, is typically appropriate.

When SPLIT MAJ X is set (this is the default), the command SPLIT GAP <nc> <separation> changes the row spacing within column <nc>. If SPLIT MAJ Y is set, however, the command SPLIT GAP <nc> <separation> changes the column spacing within row <nc>.

The command SPLIT LOCATION <x1> <x2> <y1> <y2> is analogous to the `LOCATION` command (Section 2.29) for drawing areas.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> SPLIT 3`

   Three drawing area are created, arranged in a single column.

2. `chips> SPLIT 3 2`

   Three columns are created, each with 2 rows of drawing areas. The following command is equivalent:

   ```
   chips> SPLIT COLS 2 2 2
   ```

3. `chips> SPLIT COLS 1 2 3`

   This command creates 3 columns. The first column contains one row of drawing areas, the second contains two rows, and the third column contains three rows.

4. ```
   chips> SPLIT COLS 4 1 2 3
   chips> SPLIT MAJ Y
   ```

   The first command creates four columns. The first column contains four rows of drawing areas, the second contains one, the third contains two, and the fourth contains three. The command SPLIT MAJ Y reverses the columns to rows, so that there are now four rows. The first row contains four columns, the second row contains one column, the third row contains 2 columns, and the fourth row contains three columns.

5. ```
   chips> SPLIT 3
   chips> SPLIT GAP 0.15
   ```

   Three drawing area rows are created, arranged in one column. The SPLIT GAP 0.15 command changes the gap spacing between the rows to 0.15. The following command is equivalent:

   ```
   chips> SPLIT GAP Y 0.15
   ```

6. ```
   chips> SPLIT 3 2
   chips> SPLIT GAP Y 0.12
   chips> SPLIT GAP X 0.10
   chips> SPLIT GAP 0.08
   ```

   First, three columns of drawing areas are created, each containing 2 rows. The gap spacing between the rows is changed to 0.12, and the spacing between the columns is changed to 0.10. The final command changes all gap spacings to 0.08.

7. ```
   chips> SPLIT MAJ X
   chips> SPLIT GAP 2 0.19
   ```

   The row spacing in column 2 only is changed to 0.19.

8. ```
   chips> SPLIT 3 2
   chips> SPLIT GAP 0.03
   chips> SPLIT MAJ Y
   chips> SPLIT GAP 1 0.2
   ```

   First, three columns of drawing areas are created, each containing 2 rows. All gap spacings are then set to 0.03. The SPLIT MAJ Y command reverses the column and row precedences; this changes the drawing area to two columns with three rows. The final command changes the column spacing within row one to 0.2.
   ```

2.44 STORE

Saves a plot with all attributes.

```
chips> STORE <filename>
```

The `STORE` command creates an ASCII file which contains all the commands needed to re-create the current plot display. The `<filename>` is the name of the file to be created and may contain a directory path. This file can then be read into another *ChIPS* session using the `RESTORE` command (Section 2.41).

This allows you to create a plot - perhaps in *Sherpa* - and save it so it can be re-created or modified at a later date.

When storing plots created using the *ChIPS* `CURVE` command, the store file will reference the data file by name - including the full path. If the plots were created without an associated data file - for instance a *Sherpa* plot or data plotted using the the *S-Lang* `curve()` function - then the plot data will be written to a FITS file called "`<filename>.fits`". The one exception to this is for contour plots, when the data for that plot is stored in the `/tmp/` directory as an ASCII file.

Examples:

1. ```
   chips> cd /work
   chips> SPLIT 2
   chips> D 1 CURVE data/data.dat
   chips> D 1 CURVE data/fit.dat
   chips> D 2 CURVE data/residuals.dat
   chips> D 1 C 2 SYMBOL NONE
   chips> D 1 C 2 SIMPLELINE
   chips> D 1,2 LIMITS 0.0 11.0
   chips> D 2 SYMBOL CIRCLE
   chips> D 2 SYMBOL SIZE 1
   chips> D 1 LIMITS Y 1.0 4.5
   chips> D 2 LIMITS Y 0.0 4.5
   chips> XLABEL "Off-Axis (arcmin)"
   chips> D 1,2 YLABEL "SNR"
   chips> TITLE "HRC-I 100ks"
   chips> TITLE CYAN
   chips> XLABEL CYAN
   chips> D 1,2 YLABEL CYAN
   chips> SPLIT GAP y 0.04
   chips> D 1 TICKVALS X OFF
   chips> D 1,2 TICKVALS y "%1.1f"
   chips> STORE myplot.chp
   ```

   After plotting some data and customizing the look of the plot, it is stored to the file `myplot.chp` in the current directory. This is an ASCII file the references the plotted data using full path specifications; it may be restored from any location, as long as the data has not been moved.

   ```
   chips> $more myplot.chp
   ```
   ```

```
#This is a ChIPS state file.
#ChIPS_Version: 2.20

redraw off

##############################
# Start of a new drawing area.
##############################
drawarea 0.15 0.9 0.52 0.9

#######################
# Start of a new curve.
skip 0
curve "/work/data/data.dat"

#######################
# Start of a new curve.
skip 0
curve "/work/data/fit.dat"
symbol  none
symbol 0 0
 simpleline

(etc.)
```

2. ```
   chips> x = [0:10:0.02]
   chips> y = tan(x)
   chips> chips.curvestyle = _chips->simpleline
   chips> chips.symbolstyle = _chips->none
   chips> () = curve(x,y)
   chips> limits y -10 10
   chips> store tanplot.chp
   ```

   Here we use *S-Lang* commands to create and plot the curve `y=tan(x)`. As well as creating the file `tanplot.chp`, the `STORE` command also creates the file `tanplot.chp.fits` which contains the data that was plotted.

   Note that the store file does not record the contents of the *ChIPS* State object. So, when the plot is re-created via a call to `RESTORE`, it will be drawn with no symbols and with a line connecting the points but the `chips.curvestyle` and `chips.symbolstyle` fields will not be set to `_chips->simpleline` and `_chips->none` respectively.

3. ```
   sherpa> READ DATA 1 data/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
   sherpa> LPLOT FIT
   sherpa> STORE myplotfromsherpa.chp
   sherpa> ls myplotfromsherpa.chp*
   myplotfromsherpa.chp
   ```

`myplotfromsherpa.chp.fits`

Here the `STORE` command is used from within Sherpa to save a plot generated using the Sherpa command `LPLOT`. Two files have been generated: the ASCII file with the plot commands and a FITS file that contains the data. The data for each curve is stored in a separate FITS extension:

```
chips> list myplotfromsherpa.chp.fits blocks

---------------------------------------------------------------------
Dataset: myplotfromsherpa.chp.fits
---------------------------------------------------------------------

     Block Name                          Type         Dimensions
---------------------------------------------------------------------
Block    1: PRIMARY                        Null
Block    2: curve0                         Table         6 cols x 12
Block    3: curve1                         Table         2 cols x 12
```

The [`curve0`] extension contains the input data and the [`curve1`] extension contains the fit.

2.45 SURFACE

Adds a surface plot to a drawing area.

```
chips> [D #] SURFACE <filename>[<virtual_file_syntax>] <zmin> <zmax>
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <filename>
Description: name of datafile and path, if necessary.  If the file
             contains tabulated data, it must be sorted with row
             precedence (see below).  May be filtered data from a
             FITS image file, as shown in the EXAMPLES.
```

```
Argument: <virtual_file_syntax>
Description: A filtering and/or binning command argument.  The
             <virtual_file_syntax> must be surrounded by
             quotes (" ").
```

See the `CURVE` command (Section 2.11) for more information on this argument.

```
Argument: <zmin>
Description: minimum Z-axis value
Options: real numbers
```

```
Argument: <zmax>
Description: maximum Z-axis value
Options: real numbers
```

Surface are considered variants of curves, and therefore the attributes of contours (color, linestyle, scale, width) may be changed in the same manner. However, the limits of the x and y axes in a surface plot cannot be modified.

The `VIEWPOINT` command (Section 2.53) has information on controlling the rotation angle of a surface plot.

SORTING THE INPUT DATA

The input data file must be sorted with row precedence. For example, the following data:

```
1 1 1
2 1 -4.97206e-24
3 1 2
4 1 1
5 1 2
1 2 -6.79138e-24
2 2 -2.23247e-23
3 2 1
4 2 -2.1879e-22
5 2 1
```

Would have to be re-ordered for use in *ChIPS*:

```
1 1 1
1 2 -6.79138e-24
2 1 -4.97206e-24
2 2 -2.23247e-23
3 1 2
3 2 1
4 1 1
4 2 -2.1879e-22
5 1 2
5 2 1
```

Examples:

1. `chips> SURFACE data/example3D.sorted.dat 0.0 10.0`

   The ASCII data file data/example3D.sorted.dat is plotted as a surface, where the minimum Z-axis value is 0.0 and the maximum Z-axis value is 10.0.

2. `chips> SURFACE data/exampleImage.fits[10:100,10:100] 0.0 10.0`

   A surface plot is created from the data in the FITS file which falls into the specified ranges. Again, the minimum Z-axis value is 0.0 and the maximum Z-axis value is 10.0.

3. ```
   chips> CLEAR
   chips> SURFACE "data/example3D.sorted.fits[cols col1, col2, col3]"
             0.0 10.0
   ```

   The data in columns col1, col2, and col3 of data/example3D.sorted.fits are plotted as a surface plot.

4. `chips> SURFACE s3_img.fits 20 30`

   A surface plot is created from the filtered data stored in "s3_img.fits".

2.46 SYMBOL

Specifies the appearance of symbols in a curve.

```
chips> [D #] [C #] SYMBOL <color>
chips> [D #] [C #] SYMBOL <symbolstyle>
chips> [D #] [C #] SYMBOL [SIZE] <size_value>
```

```
Argument: C #
Description: curve number designation
Options: integer numbers
Default: current curve
```

See the `C` command (Section 2.7) for more information about this argument.

```
Argument: <color>
Description: axes color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <size_value>
Description: symbol size; affects all subsequent symbol styles.
             Has no effect when the current symbol style is POINT.
Options: real numbers
Default: 3.0
```
```

```
Argument: <symbolstyle>
Description: style for symbols
Options: BLOCK, CIRCLE, CROSS, DIAMOND, SOLIDDIAMOND, DOWNTRI,
         SOLIDDOWNTRI, UPTRI, SOLIDUPTRI, SQUARE, POINT, BIGPOINT,
         NONE
Default: CROSS
```

Note the the “POINT” symbol is not scalable (it is always tiny). As an alternative, the “BIGPOINT” shape can be set to any size.

The Figure 2.4 is an illustration of the different `<symbolstyle>` options.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See “**ahelp chips**” for further details.

Examples:

1. ```
   chips> CURVE data/exampleA.dat
   chips> CURVE data/exampleB.dat
   chips> C 1 SYMBOL UPTRI
   chips> C 2 SYMBOL DIAMOND
   ```
   First, two curves are plotted into a single drawing area. Curve 1 is changed to UPTRI and curve 2 is changed to DIAMOND.

2. ```
   chips> C 1 SYMBOL CYAN
   chips> C 2 SYMBOL RED
   ```
   Curves 1 and 2 are changed to be cyan and red, respectively.

3. ```
   chips> C 1,2 SYMBOL SIZE 2.0
   ```
   The symbol size for both curves is increased to 2.0.

4. ```
   chips> CLEAR
   chips> CURVE data/exampleA.dat
   chips> SYMBOL DIAMOND
   chips> SYMBOL DOWNTRI
   chips> SYMBOL SIZE 2.0
   chips> SYMBOL DIAMOND
   ```
   The first two SYMBOL commands illustrate the DIAMOND and DOWNTRI symbols at their default size of 3.0. The SYMBOL SIZE 2.0 command then changes the size of the current DOWNTRI symbols. The final SYMBOL DIAMOND command shows that changing the new style retains the last SYMBOL SIZE instruction.

2.47 TICKS

Specifies the spacing of tick marks.

```
chips> [D #] TICKS [<ticktype>] [<coor>] <interval>
```

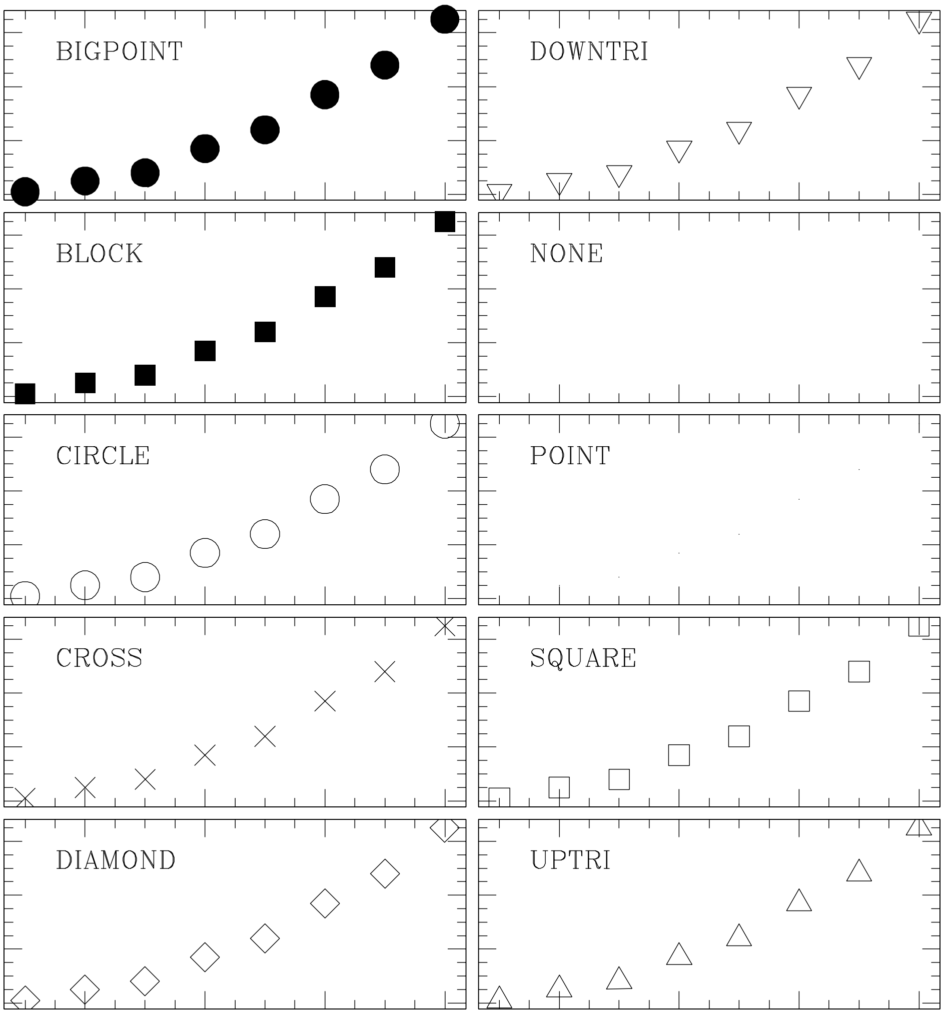

Figure 2.4: <`symbolstyle`> Options

```
Argument: <coor>
Description: axis
Options: X, Y
Default: both X and Y
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <interval>
Description: interval between tick marks
Options: real numbers
Default: automatically determined
```

```
Argument: <ticktype>
Description: type of grid markings
Options: MAJ - major tick marks only
         MIN - minor tick marks only
Default: both MAJ and MIN
```

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**`ahelp chips`**" for further details.

Examples:

1. ```
   chips> SPLIT 2
   chips> D 1 CURVE data/fit.dat
   chips> D 2 CURVE data/residuals.dat
   chips> D 1 TICKS 1.0
   ```
   The curves are plotted in two drawing areas, then the tick spacing of drawing area 1 is changed to 1.0 for both axes.

2. ```
   chips> D 2 TICKS X 2.0
   chips> D 2 TICKS Y 0.5
   ```
   The first command changes the interval between tick marks on the X-axis of drawing area 2 to 2.0. The Y-axis tick marks are changed to have a spacing of 0.5.

3. ```
   chips> SPLIT GAP 0.05
   chips> D 1 TICKS MIN X 0.5
   chips> D 1 TICKS MAJ Y 0.5
   chips> D 2 TICKS MAJ X 1.0
   chips> D 2 TICKS MIN X 0.5
   ```
   The drawing areas are first separated by a small amount (0.05). The command D 1 TICKS MIN X 0.5 adds minor tick marks with spacing 0.5, to the X-axis of drawing area 1. The interval between the Y-axis major tick marks of that area are then changed to 0.5. Similar commands are applied to drawing area 2.
   ```

2.48 TICKVALS

Controls the appearance of numerical tick mark labels.

```
chips> [D #] TICKVALS [<coor>] {AUTO | <format>}
chips> [D #] TICKVALS [<coor>] {ON | OFF}
chips> [D #] TICKVALS [SIZE] <size_value>
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <coor>
Description: axis
Options: X, Y
Default: both X and Y
```

```
Argument: <format>
Description: format of the numeric tick mark labels; must be contained
             within quotes (" ").
Options: any standard FORTRAN or C format string
```

```
Argument: <size_value>
Description: tick size
Options: real numbers
Default: 1.0
```

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. ```
   chips> SPLIT 2
   chips> D 1 CURVE data/fit.dat
   chips> D 2 CURVE data/residuals.dat
   chips> D 1 TICKVALS X OFF
   ```
   After two curves are plotted, each within a separate drawing area, the tick mark labels are removed from the X-axis of drawing area number 1.

2. ```
   chips> D 1 TICKVALS SIZE 0.95
   chips> D 2 TICKVALS SIZE 0.75
   ```
   The size of the tick marks on drawing areas 1 and 2 are changed to 0.95 and 0.75, respectively.

3. `chips> D 1 TICKVALS Y "%1.1f"`
   `chips> D 2 TICKVALS "%1.1f"`
   `chips> D 2 TICKVALS X AUTO`

   The format of the tick mark labels on the Y axis of drawing area 1, where "%1.1f" is a C formatting string. The second command applies the same format to both axes of drawing area 2. Finally, the tick mark labels on the X-axis of area 2 are restored to the default setting.

2.49 TITLE

Adds a title and/or changes the title attribute.

```
chips> TITLE <color>
chips> TITLE ["<title_text>"]
chips> TITLE SIZE ["<size_value>"]
```

```
Argument: <color>
Description: title color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <title_text>
Description: the text for the title, which must be in quotes
             (" ").  The <text> may contain TeX syntax for
             formatting text, creating mathematical expressions,
             and mixing font settings.
Options: quoted character string
```

The `FONT` command (Section 2.18) has more information about font options.

```
Argument: <size_value>
Description: title size
Options: real numbers
Default: 1.0
```

Issuing this command with no argument will delete the current title.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "`ahelp chips`" for further details.

Examples:

1. `chips> D 1 CURVE data/example.dat X 1 Y 2`

```
chips> TITLE "My Title"
chips> TITLE YELLOW
```

A data file is plotted and a title is added to a plot. The color of the title is then changed to yellow.

2. `chips> TITLE "x = \frac{y+z/2}{y^{2}+3}"`

   A title that contains a mathematical expression is added to a plot by using TeX syntax.

3. `chips> D 1 LABEL 2.0 50.0 "\oe This \rm is \gr Greek"`

   This LABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where "This" is in Old English font, "is" is in Roman font, and "Greek" is in the Greek font.

2.50 TYPE

Reports the data that are associated with a curve.

```
chips> [D #] [C #] TYPE
```

where D # is a drawing area number designation and C # is a curve number designation.

See the `C` (Section 2.7) and `D` (Section 2.12) commands for more information about these arguments.

Examples:

1.
```
chips> D 1 CURVE data/exampleA.dat
chips> D 1 CURVE data/exampleB.dat
chips> D 1 C 1 TYPE
              X              Y
              1              1
              2              5
              3              8
              4             17
              5             24
              6             37
              7             48
              8             65
chips>
```

   Two curves are plotted in drawing area 1. The TYPE command reports the data that are associated with curve number 1.

2.51 UNDO

Undoes the most recently executed plotting command.

```
chips> UNDO
chips> UNDO n
```

The `UNDO` command undoes the previous *ChIPS* command or - if a number is supplied ("`m`" above) - undoes that number of commands. There is a limit of 10 commands that can be undone.

Examples:

1. `chips> CURVE data/example.dat`
   `chips> LIMITS X 5.0 15.0`
   `chips> UNDO`

   A curve is plotted, then the limits of the X-axis are changed. The UNDO command restores the limit to their original defaults.

2. `chips> CURVE data/example.dat`
   `chips> LIMITS X 5.0 15.0`
   `chips> LIMITS Y 10.0 15.0`
   `chips> AXES BLUE`
   `chips> LIMITS Y 10.0 25.0`
   `chips> UNDO 3`

   A curve is plotted and the limits of both axes are changed. They are colored blue, then the Y-axis limits are changed again. The `UNDO` command (Section 2.51) backtracks through the previous three commands (in the order they are undone: LIMITS Y 10.0 25.0, AXES BLUE, LIMITS Y 10.0 15.0).

2.52 VERBOSE

Regulates reporting of error and informational messages.

```
chips> VERBOSE <integer>
```

where <integer> is 1 for maximum reporting and 0 for reduced reporting.

By default, maximum reporting is turned on.

Examples:

1. `chips> VERBOSE 0`

   The verbosity is reduced for this session.

2.53 VIEWPOINT

Controls the rotation angle of a surface plot.

```
chips> [D #] [C #] VIEWPOINT <theta> <phi> <l>
```

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: C #
Description: curve number designation
Options: integer number
Default: current curve
```

See the `C` command (Section 2.7) for more information about this argument.

```
Argument: <l>
Description: angle l in degrees
Options: real numbers
Default: -1.0
```

```
Argument: <phi>
Description: angle phi in degrees
Options: real numbers
Default: -10.0
```

```
Argument: <theta>
Description: angle theta in degrees
Options: real numbers
Default: 30.0
```

The angles <theta>, <phi>, and <l> are defined in *ChIPS* the same way as they are defined in the SM engine: Surfaces are drawn from a direction (<theta>,<phi>), and projected onto a surface passing through the origin. The projection is from a point <l> away from the nearest corner of the cube containing the image. If <l> is:

- positive - a perspective projection is used
- zero (0) - the viewpoint is taken to be infinitely far from the surface
- negative - an axonometric projection is used (*i.e.* the surface is projected from infinity onto the X-Z plane)

The coordinate system is right-handed and is oriented such that the Z-axis is <theta> = 90 and (<theta>,<phi>)=(0,0). Angles are taken to be in degrees, with theta lying in (-90,90) and <phi> lying in (-180,180). The nearest corner of the cube containing the surface is projected onto the point (0,0).

Examples:

1. `chips> D 1 SURFACE data/example3D.sorted.dat 0.0 10.0`
   `chips> VIEWPOINT 10.0 10.0 10.0`

   The data file data/example3D.sorted.dat is plotted as a surface plot. The VIEWPOINT command then changes the rotation angle and distance at which the surface plot is viewed.

2.54 XLABEL

Adds a label along the X-axis of a drawing area and/or changes X-axis label attributes.

```
chips> [D #] XLABEL ["<text>"]
```

To change X-axis label attributes:

```
chips> [D #] XLABEL <color>
chips> [D #] XLABEL [SIZE] <size_value>
```

The XLABEL command can take many arguments:

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <color>
Description: label color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <size_value>
Description: label size
Options: real numbers
Default: 1.0
```

```
Argument: <text>
Description: the text for the label, which must be in quotes
             (" ").  The <text> may contain TeX syntax for
```

```
             formatting text, creating mathematical expressions,
             and mixing font settings.
Options: quoted character string
```

The `FONT` command (Section 2.18) has more information about font options.

Issuing this command with no arguments will delete the current X-axis label.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> CURVE data/example.dat`
   `chips> XLABEL "This is the X-axis"`

   After a curve has been plotted, a label is added to the X-axis.

2. `chips> XLABEL " "`
   `chips> XLABEL "\oe This \rm is \gr Greek"`

   The command `XLABEL " "` removes the previous X-axis label. The second XLABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where "This" is in Old English font, "is" is in Roman font, and "Greek" is in the Greek font.

3. `chips> XLABEL YELLOW`
   `chips> XLABEL SIZE 1.5`

   An existing label is changed to the color yellow, then increased in size to 1.5.

2.55 YLABEL

Adds a label along the Y-axis of a drawing area and/or changes Y-axis label attributes.

```
chips> [D #] YLABEL ["<text>"]
```

To change Y-axis label attributes:

```
chips> [D #] YLABEL <color>
chips> [D #] YLABEL [SIZE] <size_value>
```

The YLABEL command can take many arguments:

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <color>
Description: label color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <size_value>
Description: label size
Options: real numbers
Default: 1.0
```

```
Argument: <text>
Description: the text for the label, which must be in quotes
             (" ").  The <text> may contain TeX syntax for
             formatting text, creating mathematical expressions,
             and mixing font settings.
Options: quoted character string
```

The `FONT` command (Section 2.18) has more information about font options.

Issuing this command with no arguments will delete the current Y-axis label.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. `chips> CURVE data/example.dat`
   `chips> YLABEL "This is the Y-axis"`

   After a curve has been plotted, a label is added to the Y-axis.

2. `chips> YLABEL " "`
   `chips> YLABEL "\oe This \rm is \gr Greek"`

   The command YLABEL " " removes the previous Y-axis label. The second YLABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where "This" is in Old English font, "is" is in Roman font, and "Greek" is in the Greek font.

3. `chips> YLABEL YELLOW`
   `chips> YLABEL SIZE 1.5`

   An existing label is changed to the color yellow, then increased in size to 1.5.

2.56 ZLABEL

Adds a label along the Z-axis of a drawing area and/or changes Z-axis label attributes.

```
chips> [D #] ZLABEL ["<text>"]
```

To change Z-axis label attributes:

```
chips> [D #] ZLABEL <color>
chips> [D #] ZLABEL [SIZE] <size_value>
```

The ZLABEL command can take many arguments:

```
Argument: D #
Description: drawing area number designation
Options: integer numbers
Default: current drawing area
```

See the `D` command (Section 2.12) for more information about this argument.

```
Argument: <color>
Description: label color
Options: BLACK, BLUE, CYAN, DEFAULT, GREEN, MAGENTA, RED, WHITE,
         YELLOW
Default: DEFAULT (appears in ChIPS window as white; prints as black)
```

```
Argument: <size_value>
Description: label size
Options: real numbers
Default: 1.0
```

```
Argument: <text>
Description: the text for the label, which must be in quotes
             (" ").  The <text> may contain TeX syntax for
             formatting text, creating mathematical expressions,
             and mixing font settings.
Options: quoted character string
```

The `FONT` command (Section 2.18) has more information about font options.

Issuing this command with no arguments will delete the current Z-axis label.

CIAO allows some plot attributes to be changed using the *ChIPS* state object and resource file. See "**ahelp chips**" for further details.

Examples:

1. ```
   chips> CURVE data/example.dat
   chips> ZLABEL "This is the Z-axis"
   ```
   After a curve has been plotted, a label is added to the Z-axis.
   ```

2. `chips> ZLABEL " "`
   `chips> ZLABEL "\oe This \rm is \gr Greek"`

   The command ZLABEL " " removes the previous Z-axis label. The second ZLABEL command utilizes TeX syntax in order to achieve mixed font settings in the same plot. The command produces a label where "This" is in Old English font, "is" is in Roman font, and "Greek" is in the Greek font.

3. `chips> ZLABEL YELLOW`
   `chips> ZLABEL SIZE 1.5`

   An existing label is changed to the color yellow, then increased in size to 1.5.

Chapter 3

S-lang in *ChIPS*

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 *ChIPS* commands.

All of the information provided here is available within CIAO and *ChIPS* from the **ahelp** system; "**ahelp -c chips**" returns a list of all the help files relating to *ChIPS* .

Unlike *ChIPS* commands, S-lang **IS case sensitive** and must be written in lowercase.

3.1 chips_auto_redraw

Set the automatic redrawing mode.

```
Integer_Type chips_auto_redraw(flag)
```

This function sets (flag = 1) or unsets (flag = 0) the automatic redrawing mode of *ChIPS*. It returns the previous setting.

Use the "chips_redraw()" command to force the *ChIPS* display to be redrawn.

Name	Type	Options	Default	Comment
flag	Integer_Type	0 for off, 1 for on	1 (on)	

Table 3.1: Arguments for chips_auto_redraw

Examples:

1. `chips> chips_auto_redraw(0)`

```
1
```

Here we have turned off the automatic redraw mode. The return value shows that the previous setting had been on (*i.e.* 1).

2. `() = chips_auto_redraw(0);`

   As the first example, but done in a S-Lang script. Here the return value is ignored.

3.2 chips_clear

Removes all plotting objects, creating a new blank drawing area.

```
chips_clear()
```

This is a S-Lang version of the *ChIPS* `CLEAR` command (Section 2.8). It erases all curves, lines, labels, and resets the number of panes (*i.e.* drawing areas) to 1. Since it has no parameters, it can be called without the "()".

Examples:

1. `chips> chips_clear`

   Clear *ChIPS* of all plotting objects.

2. `() = chips_clear;`

   As the first example, but done in a S-Lang script.

3.3 chips_color_name

Converts a color number to a string.

```
String_Type chips_color_name([colorValue])
```

Returns the string equivalent of the symbolic color values in the "_chips" namespace. The values are described in the 'Attribute values' section of "ahelp chips". The "chips_color_value()" function can be used to convert the names of colors to their symbolic value.

Name	Type	Options	Default	Comment
colorValue	Integer_Type	black, blue, cyan, default, green, magenta, red, white, yellow (see Example 3)	default	Optional

Table 3.2: Arguments for chips_color_name

If colorValue is outside the range of supported colors, then the string "default" will be returned. If the color value is omitted the routine will return a string containing all the available colors, separated by a newline (*i.e.* '\n') character.

Examples:

1. ```
   chips> chips_color_name( _chips->red )
   red
   ```

   Here we convert the value _chips->red to the string "red". Since we have ignored the return value, *ChIPS* prints it out to the screen (see the 'Using ChIPS and Sherpa as a calculator' section of "ahelp tips").

2. ```
   chips> variable colname = chips_color_name( chips.curvecolor );
   chips> print("The color of curves is " + colname );
   ```

   Here use use S-Lang code to find out the name of the color used to draw curves. If the *ChIPS* state object has not been changed, then these commands would produce the following output:

   ```
   The color of curves is default
   ```

3. ```
   chips> chips_color_name()
   black
   blue
   cyan
   default
   green
   magenta
   red
   white
   yellow
   ```

   Since no value is given, the list of available colors is returned. This string may be stored in a variable:

   ```
   chips> hues = chips_color_name()
   chips> print(hues)
   black
   blue
   cyan
   default
   green
   magenta
   red
   white
   yellow
   ```

   As the names are separated by the newline ('\n') character, they can be easily separated using the S-Lang strchop() function:

   ```
   chips> colors = strchop( chips_color_name(), '\n', 0 )
   chips> print(colors[2])
   cyan
   ```
   ```

3.4 chips_color_value

Converts the name of a color to its numeric value.

```
Integer_Type chips_color_value(colorName)
```

Returns the symbolic value for the supplied color name. The color values are described in the 'Attribute values' section of "ahelp chips". The "chips_color_name()" function can be used to convert the symbolic value back to the name of a color.

Name	Type	Default	Comment
colorName	String_Type	3	Case insensitive

Table 3.3: Arguments for chips_color_value

The value of _chips->default will be returned when an unknown color is specified.

Examples:

1.
```
chips> chips_color_value( "red" )
6
chips> _chips->red
6
```
Here we find the symbolic value for the color "red". Since we have ignored the return value *ChIPS* prints it out to the screen (see the 'Using ChIPS and Sherpa as a calculator' section of "ahelp tips").

2.
```
chips> chips_color_value( "green" )
4
chips> chips_color_value( "GREEN" )
4
chips> chips_color_value( "GrEEn" )
4
```
The case of the color name is unimportant, as shown here.

3.
```
chips> chips_color_value( "UnknownColor" )
3
chips> chips_color_value( "default" )
3
```
If a color is unknown, then the value for the default color is returned.

4.
```
chips> chips.curvecolor = chips_color_value( "red" );
```
Here we set the curvecolor attribute of the *ChIPS* state object (see "ahelp chips") to be red. Since curvecolor must be specified using the symbolic value for the color red, we use chips_color_value() to do the conversion. The following line of code can also be used to set the curvecolor attribute to red:
```
chips> chips.curvecolor = _chips->red;
```

3.5 chips_get_pane

Get the current pane/drawing area.

```
Integer_Type chips_get_pane()
Integer_Type chips_get_drawing_area()
```

Returns the number of the current pane/drawing area; the two commands are identical. The numbering scheme of the panes is described in "ahelp chips_split". Since there are no parameters, the functions can be called without the trailing "()".

Examples:

1.
```
chips> chips_split(2,3,4)
6
chips> chips_get_pane
4
```
   Here we create 6 panes, arranged in a 2 by 3 grid, and move to the fourth one. We then check that this has happened with chips_get_pane().

2.
```
() = chips_split(2,3,4);
variable pane = chips_get_pane;
print("Current pane is " + string(pane))
```
   As the first example, but done in a S-Lang script. Here the pane number is stored in a S-Lang variable and output to the screen using the Varmm print() function.

3.6 chips_get_xrange

Get the upper and/or lower x-axis limits of the plot

```
Integer_Type chips_get_xrange()
```

Determines the limits of the plot's x-axis. On success, two float values will be returned. Two NULLs are returned on error (*e.g.* if there is no drawing area). Note that the limits returned are those used internally by *ChIPS* to draw the plot axes, so they may not match the numeric limits of the plotted data.

To return either the minimum or the maximum value, the syntax

```
(xmin,) = chips_get_xrange()
(,xmax) = chips_get_xrange()
```

can be used.

The limits may be adjusted with **chips_set_xrange** (Section 3.17).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> (xmin,xmax) = chips_get_xrange()
chips> vmessage("xmin = %f, xmax = %f",xmin,xmax)
xmin = 751.849976, xmax = 887.150024
```
A data file is read in and plotted, then the minimum and maximum values of the x-axis are printed out.

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> (,rmax) = chips_get_xrange()
chips> print(rmax)
72.45
```
Here only the upper (maximum) x-axis limit was desired.

3.7 chips_get_xscale

Get the x-axis scale

```
Integer_Type chips_get_xscale()
```

Gets the scale of the plot's x-axis. The function returns "1" for linear and "0" for log. Non-existent drawing areas are assumed to have a default scale of linear.

The current scale may be changed with **chips_set_xscale** (Section 3.18).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_get_xscale()
1
```
The dataset is read in and plotted with a linear x-axis, as returned by the "chips_get_xscale" command.

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_xscale(0)
Warning: negative and zero values ignored in log scale
0
chips> chips_get_xscale()
0
```
The x-axis is changed to log scale with the "chips_set_xscale" command, then the change is confirmed. Note the the first command returns "0" for success, while the second returns "0" to indicate log scale.

3.8 chips_get_yrange

Get the upper and/or lower y-axis limits of the plot

```
Integer_Type chips_get_yrange()
```

Determines the limits of the plot's y-axis. On success, two float values will be returned. Two NULLs are returned on error (*e.g.* if there is no drawing area). Note that the limits returned are those used internally by *ChIPS* to draw the plot axes, so they may not match the numeric limits of the plotted data.

To return either the minimum or the maximum value, the syntax

```
(ymin,) = chips_get_yrange()
(,ymax) = chips_get_yrange()
```

can be used.

The limits may be adjusted with `chips_set_yrange` (Section 3.19).

Examples:

1. ```
   chips> input=readfile("/data/chips/phas.fits")
   chips> curve x input.x y input.y
   chips> (ymin,ymax) = chips_get_yrange()
   chips> vmessage("ymin = %f, ymax = %f",ymin,ymax)
   ymin = -6.300000, ymax = 132.300003
   ```

   A data file is read in and plotted, then the minimum and maximum values of the y-axis are printed out.

2. ```
   chips> spec=readfile("/data/threads/Chips/data1.pha")
   chips> curve x spec.channels y spec.counts
   chips> (,rmax) = chips_get_yrange()
   chips> print(rmax)
   36.75
   ```

   Here only the upper (maximum) y-axis limit was desired.

3.9 chips_get_yscale

Get the y-axis scale

```
Integer_Type chips_get_yscale()
```

Gets the scale of the plot's y-axis. The function returns "1" for linear and "0" for log. Non-existent drawing areas are assumed to have a default scale of linear.

The current scale may be changed with `chips_set_yscale` (Section 3.20).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_get_yscale()
1
```
   The dataset is read in and plotted with a linear y-axis, as returned by the "chips_get_yscale" command.

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_yscale(0)
Warning: negative and zero values ignored in log scale
0
chips> chips_get_yscale()
0
```
   The y-axis is changed to log scale with the "chips_set_yscale" command, then the change is confirmed. Note the the first command returns "0" for success, while the second returns "0" to indicate log scale.

3.10 chips_get_zrange

Get the upper and/or lower z-axis limits of the plot

```
Integer_Type chips_get_zrange()
```

Determines the limits of the plot's z-axis. On success, two float values will be returned. Two NULLs are returned on error (*e.g.* if there is no drawing area). Note that the limits returned are those used internally by *ChIPS* to draw the plot axes, so they may not match the numeric limits of the plotted data.

To return either the minimum or the maximum value, the syntax

```
(zmin,) = chips_get_zrange()
(,zmax) = chips_get_zrange()
```

can be used.

The limits may be adjusted with `chips_set_zrange` (Section 3.21).

Examples:

1. ```
   chips> surface tabbed.data 0 150
   chips> (zmin,zmax) = chips_get_zrange()
   chips> vmessage("zmin = %f, zmax = %f",zmin,zmax)
   zmin = 0.000000, zmax = 150.000000
   ```
   The minimum and maximum values of the z-axis for a surface plot are printed out.

2. ```
   chips> surface tabbed.data 0 100
   chips> (,rmax) = chips_get_zrange()
   chips> print(rmax)
   100
   ```
   Here only the upper (maximum) z-axis limit was desired.

3.11 chips_get_zscale

Get the z-axis scale

```
Integer_Type chips_get_zscale()
```

Gets the scale of the plot's z-axis. The function returns "1" for linear and "0" for log. Non-existent drawing areas are assumed to have a default scale of linear.

The current scale may be changed with `chips_set_zscale` (Section 3.22).

Examples:

1. ```
   chips> surface tabbed.data 0 150
   chips> chips_get_zscale()
   1
   ```
   The surface is plotted with a linear z-axis, as returned by the "chips_get_zscale" command.

3.12 chips_label

Adds a label to a drawing area.

```
Integer_Type chips_label(x,y,text)
Integer_Type chips_label(x,y,text,color)
Integer_Type chips_label(x,y,text,color,size)
Integer_Type chips_label(x,y,text,color,size,rotationAngle)
```

This is a S-Lang version of the *ChIPS* `LABEL` command (Section 2.22). It adds the text to the current pane at the specified location (x,y), and the optional parameters control the color, size, and rotation angle of the label.
```

Name	Type	Default	Comment
x	Numeric		location of bottom left of text
y	Numeric		location of bottom left of text
text	String_Type		may contain TeX syntax
color	Integer_Type	_chips-> defcolor	
size	Numeric	1.0	
rotationAngle	Numeric	0	degrees

Table 3.4: Arguments for chips_label

The rotationAngle parameter specifies the counter-clockwise rotation of the text from the horizontal. The rotation occurs about the point specified by the x and y parameters.

The function returns a 0 on success and -1 on failure. If used with the wrong number of arguments, it prints a usage message:

```
chips> chips_label(1)
Usage: chips_label(x,y,text[,color[,rotationAngle,[size]]])
chips_label(1);
```

3.12.1 Quoting the text

Since the backslash character ('\') is used in S-Lang as a delimiter (as do printf and related functions in the C/C++ stdio library), remember to use a double backslash for embedded Tex commands or font specifications. For example,

```
chips> chips_label(0.5,0.5,"\\it {HELLO}")
```

will draw the label "HELLO" in italic when SM is the underlying plot engine. Similarly, the command

```
chips> chips_label(0.2,0.2,"\"myLabel\"")
```

ensures that the label will be enclosed by double quotes. Make sure that both sets of quotes are closed at the end.

Examples:

1.
```
chips> clear
chips> chips_label(0.5,0.5,"A label")
0
```

   Here we mix *ChIPS* and S-Lang commands. The CLEAR command is used to remove any previous plots and set the axis limits to 0 and 1. Then we use the S-Lang command to add a label to the plot at (0.5,0.5).

2.
```
chips> chips_label(0.7,0.7,"A label",_chips->red,2,180)
0
```
This time the label is placed upside-down (*i.e.* rotated by 180), written in red, and has a size of 2.

3.
```
chips_clear();
() = chips_label(0.2,0.5, "x = \\frac{y+z/2}{y^{2}+3}");
```
This part of a S-lang script would clear the screen and then add the label to it. Note the use of a double backslash, as discussed in the "Quoting the text" section, above.

4.
```
chips> () = chips_label(0.2,0.2,"\\oe This \\rm is \\gr Greek");
```
Here we use more TeX syntax in order to achieve mixed fonts in the same label. This command produces a label where the "This" text is in Old English font, the "is" text is in Roman font, and the "Greek" text is in the Greek font. See the *ChIPS* `FONT` command (Section 2.18) for more information on the available fonts.

3.13 chips_line

Adds a line to a drawing area.

```
Integer_Type chips_line(x1,y1,x2,y2)
Integer_Type chips_line(x1,y1,x2,y2,color)
Integer_Type chips_line(x1,y1,x2,y2,color,width)
Integer_Type chips_line(x1,y1,x2,y2,color,width,style)
```

This is a S-Lang version of the *ChIPS* `LINE` command (Section 2.25). It draws a line from (x1,y1) to (x2,y2) in the current pane, and the optional parameters control the color, width, and linestyle. If a parameter is not specified, the value from the *ChIPS* state object is used instead.

The function returns a 0 on success and -1 on failure.

Name	Type	Default	Comment
x1	Numeric		start of line
y1	Numeric		start of line
x2	Numeric		end of line
y2	Numeric		end of line
color	Integer_Type	chips.linecolor	
width	Numeric	chips.linewidth	
style	Numeric	chips.linestyle	

Table 3.5: Arguments for chips_line

The routine prints a usage message if used with the wrong number of arguments:

```
chips> chips_line(1)
Usage: chips_line(x1,y1,x2,y2[,color[,width[,style]]])
```

```
chips_line(1);
```

Examples:

1.
```
chips> clear
chips> set_state_defaults("chips")
chips> chips_line(0,0,2,1)
0
```

Here we mix *ChIPS* and S-Lang commands. The CLEAR command is used to remove any previous plots and set the axis limits to 0 and 1. Then we use S-Lang commands to reset the values in the *ChIPS* state object and then to draw a line between (0,0) and (2,1).

2.
```
chips> clear
chips> chips_line(0.1,0.1,0.9,0.1,_chips->red,1,_chips->dotdash)
0
```

This draws a horizontal red line, using the DOTDASH style and with a width of 1, between (0.1,0.1) and (0.9,0.1).

3.
```
chips_clear();
chips.linecolor = _chips->red;
chips.linewidth = 1;
chips.linestyle = _chips->dotdash;
() = chips_line(0.1,0.1,0.9,0.1);
```

This part of a S-lang script has the same result as the previous example. This time the *ChIPS* state object is used to set the line parameters, rather than supplying them as optional parameters to the "chips_line()" function. Note that any following lines will also be drawn in red using the DOTDASH style until the *ChIPS* state object is changed, or the attributes are explicitly specified in the chips_line() call.

3.14 chips_pickpoints

Read 1 or more cursor positions from ChIPS.

```
Float_Type chips_pickpoints( [numpoints] )
```

This is the S-Lang version of the `PICKPOINTS` command. It allows a user to select 1 or more positions on a plot and returns the values in an array. The number of points is determined by the optional parameter `numpoints`; if not supplied it defaults to 1, otherwise it must be an integer with value greater than 0. On error, the routine returns `NULL`.

The return value is a one-dimensional array of floats where the selected points are stored in (x_i, y_i) order. The routines reshape() and _reshape() from the S-Lang Run-Time Library can be used to change the format of the data as shown in the examples below; see "ahelp reshape" and "ahelp _reshape" for more information.

Examples:

1. ```
   chips> clear
   chips> limits 0 10 0 5
   chips> pnts = chips_pickpoints

   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (5.656155, 4.306668)
   chips> print(pnts)
   5.65615
   4.30667
   chips> print(pnts[0])
   5.65615
   chips> print(pnts[1])
   4.30667
   chips> typeof(pnts)
   Array_Type
   chips> _typeof(pnts)
   Float_Type
   ```

   Here we create a plot with limits 0-10 and 0-5 on the X and Y axes and then call the S-Lang version of `PICKPOINTS`. Since we only wanted to select one point, we did not supply an argument - or "()"'s - to the `chips_pickpoints()` call.

   The screen output and behaviour is the same as if `PICKPOINTS` (Section 2.33) had been used. The difference is that the selected values are returned as a one-dimensional array; the variable `pnts` is used to store the data.

2. ```
   chips> clear
   chips> limits 0 10 0 5
   chips> pnts = chips_pickpoints(3)

   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (8.651068, 4.245632)
   Point picked: (5.343642, 2.890631)
   Point picked: (5.031129, 2.133974)
   chips> print(pnts)
   8.65107
   4.24563
   5.34364
   2.89063
   5.03113
   2.13397
   chips> vmessage("Point 1 = %f,%f",pnts[0],pnts[1])
   Point 1 = 8.651068,4.245632
   chips> vmessage("Point 2 = %f,%f",pnts[2],pnts[3])
   Point 2 = 5.343642,2.890631
   chips> vmessage("Point 3 = %f,%f",pnts[4],pnts[5])
   Point 3 = 5.031129,2.133974
   ```

   Here we ask for three points rather than 1 (and so need to use "()" around the argument). Note that the output array is still one-dimensional, with the values stored in (x_i, y_i) order.

3. ```
   chips> clear
   chips> limits 100 120 200 300
   chips> pnts = chips_pickpoints(4)

   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (106.052490, 277.100006)
   Point picked: (105.739983, 227.054245)
   Point picked: (115.010376, 237.796600)
   Point picked: (110.426857, 209.720001)
   chips> reshape(pnts,[4,2])
   chips> print(pnts)
   106.052 277.1
   105.74 227.054
   115.01 237.797
   110.427 209.72
   chips> xpnts = pnts[*,0]
   chips> print(xpnts)
   106.052
   105.74
   115.01
   110.427
   chips> ypnts = pnts[*,1]
   chips> print(ypnts)
   277.1
   227.054
   237.797
   209.72
   ```

   Here we use the `reshape()` routine to change the format of the returned array from one-dimensional - with eight elements - to a two-dimensional array with 4x2 elements.

   We then use the array-indexing capabilities of S-Lang to take a "slice" of this array to extract just the x (`xpnts`) and y (`ypnts`) values.

   Alternatively we could have used

   $$pos1 = pnts[0, *] \tag{3.1}$$

   to create a two-element array `pos1` which contains the x and y coordinates of the first point in the list.

4. ```
   chips> clear
   chips> limits 0 10 0.01 10
   chips> log y
   chips> lpnts = chips_pickpoints(2)

   Click LMB or tap spacebar to pick point.
   Click RMB or type 'q' to quit picking points.

   Point picked: (2.192879, -1.239185)
   Point picked: (9.458393, -0.487297)
   chips> print(lpnts)
   ```

```
2.19288
-1.23919
9.45839
-0.487297
chips> i=[1,3]
chips> writeascii(stdout,lpnts[i],10^lpnts[i])
-1.23919 0.0576521
-0.487297 0.325614
```

Here we use a plot in which one of the axes (the Y axis) has log spacing. The return values for that axis are still in logarithmic units (which is why they are both less than 0). The `writeascii()` call writes out the y values of the selected points in the first column, and the corresponding linear values in the second column. We take advantage of S-Lang's array-indexing mechanism to restrict the output to just use the second and fourth elements (ie "[1,3]") of the `lpnts` array.

5. The previous examples have shown `chips_pickpoints()` being used from the ChIPS prompt. It can also be used from a S-Lang script, as the following example shows.

```
unix% cat pick.sl
% load the ChIPS functions
require("chips");

% turn off redrawing until the plot is finished
() = chips_auto_redraw(0);

% set the limits
() = chips_set_xrange(0,10);
() = chips_set_yrange(10,20);

% plot some data
variable x = [1:9];
variable y = x + 10;
chips.symbolstyle = _chips->none;
chips.linestyle   = _chips->simpleline;
() = curve(x,y);
() = chips_auto_redraw(1);

% call pickpoints
variable out = chips_pickpoints(3);
if ( andelse { out == NULL }
             { length(out) != 6 } )
  error( "Did not select 3 points from the plot" );

print("\nYou selected the following values:");
variable i;
foreach ( [1:3] ) {
  i = ();
  () = printf( "  point %d  x = %4.2f  y = %5.2f\n",
               i, out[2*(i-1)], out[2*(i-1)+1] );
}
% end of script
```

which, when run using slsh, will create output something like (depending on the actual points selected):

```
unix% slsh pick.sl

Click LMB or tap spacebar to pick point.
Click RMB or type 'q' to quit picking points.

Point picked: (3.468972, 15.146487)
Point picked: (6.880163, 14.975967)
Point picked: (9.093795, 18.173878)

You selected the following values:
  point 1  x = 3.47  y = 15.15
  point 2  x = 6.88  y = 14.98
  point 3  x = 9.09  y = 18.17
```

3.15 chips_redraw

Redraws all plotting objects.

```
chips_redraw()
```

This is a S-Lang version of the *ChIPS* `REDRAW` command (Section 2.39). Since it has no parameters, it can be called without the "()".

Use the `chips_auto_redraw()` command (Section 3.1) to set/unset the automatic redrawing of plots.

Examples:

1. `chips> chips_redraw`

   Redraws everything in the plotting window.

2. `chips_redraw;`

   As the first example, but done in a S-Lang script.

3.16 chips_set_pane

Set the current pane/drawing area.

```
Integer_Type chips_set_pane(which)
Integer_Type chips_set_drawing_area(which)
```

This is a S-Lang version of the *ChIPS* `D` command (Section 2.12). It sets the current pane/drawing area; the two commands are identical. The numbering scheme of the panes is described in "`ahelp chips_split`" (see Section 3.23).

The function returns a 0 on success and -1 on failure.

Name	Type	Comment
which	Integer_Type	number of the pane

Table 3.6: Arguments for chips_set_pane

If "which" exceeds the number of panes, the function returns a -1 and the current pane is unchanged.

Examples:

1.
```
chips> chips_split(2,3)
6
chips> chips_set_pane(2)
0
```
Here we create 6 panes, arranged in a 2 by 3 grid, and set the current pane to be the second one. This example could also be done using the optional whichPane parameter of chips_split():
```
chips> chips_split(2,3,2)
0
```

2.
```
() = chips_split(2,3);
() = chips_set_pane(2);
```
As the first example, but done in a S-Lang script. Here the return values are ignored.

3.17 chips_set_xrange

Set the upper and/or lower x-axis limits of the plot

```
Integer_Type chips_set_xrange(range)
```

Sets the limits of the plot's x-axis.

Name	Type	Options	Default
range	Numeric	a pair of numeric values, "auto", or "*" (see below)	auto

Table 3.7: Arguments for chips_set_xrange

If a pair of values is used for the "range", it may either be two comma-separated values (minimum and maximum), or a single array whose first two elements will be used as minimum and maximum. The "*" character indicates no change in the current limit, while "auto" allows the underlying plotting package to select the limit value; both of these options must be quoted to work properly.

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

The function returns a 0 on success and -1 on failure.

The current limits may be found with `chips_get_xrange` (Section 3.6).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_set_xrange(749,890)
0
```
   A data file is read in and plotted, then the minimum and maximum values of the x-axis are changed.

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_xrange(20,"*")
0
chips> chips_set_xrange("auto","auto")
0
```
   First, the lower limit of the plot is changed to 20, while the original upper limit is retained. The second "chips_set_xrange" command has the plotting package determine the best limits for the plot.

3.18 chips_set_xscale

Set the x-axis scale

```
Integer_Type chips_set_xscale(scale)
```

Sets the scale of the plot's x-axis.

Name	Type	Options	Comment
scale	Numeric	0, 1, "linear", "log", _chips->log, _chips->linear	logarithm is base 10

Table 3.8: Arguments for chips_set_xscale

When invoked with no argument, the function will reset the scale to linear.

The function returns a 0 on success and -1 on failure.

The current scale may be found with `chips_get_xscale` (Section 3.7).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_set_xscale("log")
0
```

A data file is read in and plotted, then the x-axis is changed to log scale. Equivalent commands are:

```
chips> chips_set_xscale(0)
chips> chips_set_xscale(_chips->log)
```

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_xscale(0)
Warning: negative and zero values ignored in log scale
0
chips> chips_set_xscale(_chips->linear)
0
```

First, the x-axis is set to logarithmic scale. The second "chips_set_xscale" command changes it back to a linear scale; equivalent commands are:

```
chips> chips_set_xscale(1)
chips> chips_set_xscale("linear")
chips> chips_set_xscale()
```

3.19 chips_set_yrange

Set the upper and/or lower y-axis limits of the plot

```
Integer_Type chips_set_yrange(range)
```

Sets the limits of the plot's y-axis.

Name	Type	Options	Default
range	Numeric	a pair of numeric values, "auto", or "*" (see below)	auto

Table 3.9: Arguments for chips_set_yrange

If a pair of values is used for the "range", it may either be two comma-separated values (minimum and maximum), or a single array whose first two elements will be used as minimum and maximum. The "*" character indicates no change in the current limit, while "auto" allows the underlying plotting package to select the limit value; both of these options must be quoted to work properly.

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

The function returns a 0 on success and -1 on failure.

The current limits may be found with `chips_get_yrange` (Section 3.8).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_set_yrange(-10,150)
0
```
A data file is read in and plotted, then the minimum and maximum values of the y-axis are changed.

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_yrange(10,"*")
0
chips> chips_set_yrange("auto","auto")
0
```
First, the lower limit of the plot is changed to 10, while the original upper limit is retained. The second "chips_set_yrange" command has the plotting package determine the best limits for the plot.

3.20 chips_set_yscale

Set the y-axis scale

```
Integer_Type chips_set_yscale(scale)
```

Sets the scale of the plot's y-axis.

Name	Type	Options	Comment
scale	Numeric	0, 1, "linear", "log", _chips->log, _chips->linear	logarithm is base 10

Table 3.10: Arguments for chips_set_yscale

When invoked with no argument, the function will reset the scale to linear.

The function returns a 0 on success and -1 on failure.

The current scale may be found with `chips_get_yscale` (Section 3.9).

Examples:

1.
```
chips> input=readfile("/data/chips/phas.fits")
chips> curve x input.x y input.y
chips> chips_set_yscale("log")
```

```
0
```

A data file is read in and plotted, then the y-axis is changed to log scale. Equivalent commands are:

```
chips> chips_set_yscale(0)
chips> chips_set_yscale(_chips->log)
```

2.
```
chips> spec=readfile("/data/threads/Chips/data1.pha")
chips> curve x spec.channels y spec.counts
chips> chips_set_yscale(0)
Warning: negative and zero values ignored in log scale
0
chips> chips_set_yscale(_chips->linear)
0
```

First, the y-axis is set to logarithmic scale. The second "chips_set_yscale" command changes it back to a linear scale; equivalent commands are:

```
chips> chips_set_yscale(1)
chips> chips_set_yscale("linear")
chips> chips_set_yscale()
```

3.21 chips_set_zrange

Set the upper and/or lower z-axis limits of the plot

`Integer_Type chips_set_zrange(range)`

Sets the limits of the plot's z-axis.

Name	Type	Options	Default
range	Numeric	a pair of numeric values, "auto", or "*" (see below)	auto

Table 3.11: Arguments for chips_set_zrange

If a pair of values is used for the "range", it may either be two comma-separated values (minimum and maximum), or a single array whose first two elements will be used as minimum and maximum. The "*" character indicates no change in the current limit, while "auto" allows the underlying plotting package to select the limit value; both of these options must be quoted to work properly.

ChIPS converts double-precision numbers to floating-point values before plotting them, which can cause problems for values that are either too large or too small. The allowed range is approximately 1e-38 to 3e38 (for both positive and negative values).

The function returns a 0 on success and -1 on failure.

The current limits may be found with **`chips_get_zrange`** (Section 3.10).

Examples:

1. ```
   chips> surface tabbed.data 0 150
   chips> chips_set_zrange(-10,140)
   0
   ```
   The minimum and maximum values of the z-axis for a surface plot are changed.

2. ```
   chips> surface tabbed.data 0 100
   chips> chips_set_zrange(-25,"*")
   0
   chips> chips_set_zrange("auto","auto")
   0
   ```
   First, the lower limit of the plot is changed to -25, while the original upper limit is retained. The second "chips_set_yrange" command has the plotting package determine the best limits for the plot.

3.22 chips_set_zscale

Set the z-axis scale

```
Integer_Type chips_set_zscale(scale)
```

Sets the scale of the plot's z-axis.

Name	Type	Options	Comment
scale	Numeric	0, 1, "linear", "log", _chips->log, _chips->linear	logarithm is base 10

Table 3.12: Arguments for chips_set_zscale

When invoked with no argument, the function will reset the scale to linear.

The function returns a 0 on success and -1 on failure.

The current scale may be found with `chips_get_zscale` (Section 3.11).

Examples:

1. ```
   chips> chips_set_zscale("log")
   0
   ```
   The z-axis for a plot is changed to log scale. Equivalent commands are:

   ```
   chips> chips_set_zscale(0)
   chips> chips_set_zscale(_chips->log)
   ```

2. ```
   chips> chips_set_zscale(0)
   ```

```
0
chips> chips_set_zscale(_chips->linear)
0
```

First, the z-axis is set to logarithmic scale. The second "chips_set_zscale" command changes it back to a linear scale; equivalent commands are:

```
chips> chips_set_zscale(1)
chips> chips_set_zscale("linear")
chips> chips_set_zscale()
```

3.23 chips_split

Creates multiple drawing areas.

```
Integer_Type chips_split(numXpanes,numYpanes[,whichPane])
```

This is a S-Lang version of the *ChIPS* `SPLIT` command (Section 2.43). It creates multiple drawing areas (also called panes), optionally selecting which one is to be the "current" one ("whichPane"). The command splits the display up into "numXpanes" by "numYpanes" drawing areas. These areas (or panes) are numbered sequentially from the upper-left, going down the rows, so

```
chips> chips_split( 2, 3, 3 )
6
chips> chips_line( 0, 0, 1, 1 )
0
```

splits the display into 2 columns of three rows and sets the current pane to be the bottom-left one (number 3). A line is then drawn in this pane.

The function returns the number of panes actually created.

Name	Type	Default	Comment
numXpanes	Integer_Type		number of columns
numYpanes	Integer_Type		number of rows
whichPane	Integer_Type	1	

Table 3.13: Arguments for chips_split

The routine prints a usage message if used with the wrong number of arguments:

```
chips> chips_split(1)
Usage: chips_split(nx,ny[,drawing_area_selected])
```

```
chips_split(1);
```

Examples:

1. ```
   chips> chips_split(1,2)
   2
   chips> split gap 0.1
   ```

   Here we mix *ChIPS* and S-Lang commands. The chips_split() call creates two drawing areas, stacked vertically. We then use a *ChIPS* `SPLIT` command (Section 2.43) to add a gap between the two panes.

2. ```
   () = chips_split(1,2);
   () = chips_eval( "split gap 0.1" );
   ```

   This is a S-Lang version of the previous example.

3. ```
   chips_clear;
   () = chips_split(2,2,3);
   () = chips_label(0.5,0.5,"Pane: 3");
   ```

   This set of commands will clear the screen, create 4 drawing areas (ina 2 by 2 grid) and add the text "Pane: 3" to the third (top right) one.

3.24 chips_version

Report the version of *ChIPS* as a number or string.

```
Ingteger_Type _chips_version
String_Type _chips_version_string
```

The *ChIPS* version number is accessible using either the "_chips_version" or "_chips_version_string" variables. This allows scripts to cleanly check for version conflicts. The variables are read-only.

Examples:

1. ```
   chips> _chips_version
   30000
   chips> _chips_version_string
   3.0
   ```

   The version number can be accessed as either an integer or a string.