URL: http://cxc.harvard.edu/ciao3.4/chips.html
Last modified: December 2006

AHELP for CIAO 3.4

chips

Context: chips

Jump to: Description Examples CHANGES IN CIAO 3.0 COMMAND−LINE OPTIONS XPA ACCESS POINT AND COMMANDS CHANGES IN CIAO 3.2 Bugs See Also

Synopsis

Introduction to ChIPS, CIAO's plotting package.

Description

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.

This help file discusses the basics of ChIPS. The individual commands are described in separate help files; a list of which can be found using "ahelp −c chips".

Here we show − as an example − the ChIPS commands used to plot the RATE (with errors) versus TIME columns from a lightcurve (here called lc.fits), change the symbol color to red, and then add various labels to the plot. The print command creates a postscript version of the plot called lc.ps, and quit is used to exit ChIPS:

```
  chips> curve "lc.fits[cols TIME,RATE,ERROR]" yerr 3
  chips> symbol red
  chips> xlabel "Time (s)"
  chips> ylabel "Rate (/s)"
  chips> title "Lightcurve"
  chips> print postfile lc.ps
  chips> quit
```

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:

Commands allowed at the ChIPS prompt or in a ChIPS command file.

- 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.

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.

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.

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

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

```
REDRAW OFF
```

before creating the plot and then calling

```
REDRAW ON
```

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

```
REDRAW
```

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.

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.

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.

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.

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
```

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

```
ahelp /chips_/
```

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

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

```
() = chips_eval( "print postfile plot1.ps" );
```

which will create a postscript file called plot1.ps.

Customizing ChIPS

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

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.

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.

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.

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

```
chips.symbolcolor = _chips−>red;
```

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

ChIPS State Object Attribute Names and Values

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

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.

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.

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

```
() = evalfile("/home/ciaouser/slang/setup_chips.sl")
```

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".

Example 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.

Example 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.

Example 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.

Example 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.

Example 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 )
```

CHANGES IN CIAO 3.0

S−lang functions

A number of S−Lang functions have been added that duplicate and extend the ChIPS commands. Use "ahelp /chips_/" to list these new functions. The "curve()" command has been enhanced to allow X and Y error bars to be plotted.

Improved limits

When calculating the limits for a plot, ChIPS now adds a border of 5% of the total width/height of the plot to the axis. This improves the display of points at the edges of the plot. The behaviour can be controlled by new fields in the State object: "fullautolimits", "padfactor", and "lowerloglimit".

Improved control of postscript plots

The following new fields in the State object can be used to control the layout and format of postscript files generated by the PRINT command: "linewidth", "colorsys", "pagewidth", "pagelen", and "unit".

Contour plots

Contours can not be over−plotted; they could be in earlier versions but were not drawn correctly when the axes differed. The "mingridsize" field has been added to the State object to deal with interpolating data onto linear grids. See "ahelp contour" for more details.

New linewidth field

The "linewidth" field has been added to the State object to allow control over the width of lines.

New symbol types

The symbol types SOLIDDIAMOND, SOLIDUPTRI, and SOLIDDOWNTRI have been added. Corresponding entries for _chips (e.g. "_chips−>soliduptri") are also available.

Store files

The format of the ASCII files created by the STORE command have been changed slightly (the old format is still understood). The first two lines now begin with a "#" character. These files − including the old format − can now be used as ChIPS command files, so you can say

```
chips store.chp
```

to restore the plot stored in store.chp.

Control−C no longer exits ChIPS

Pressing Control−C at the ChIPS prompt will no longer cause the program to stop. It can still be used to stop a command (other than when drawing a plot).

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.

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.

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.

Bugs

See the bugs page for ChIPS on the CIAO website for an up−to−date listing of known bugs.

See Also

calibration
    caldb
chandra
    coords, guide, isis, level, pileup, times
chips
    chips_eval
concept
    autoname, configure, parameter, stack, subspace

dm
: dm, dmbinning, dmcols, dmfiltering, dmimages, dmimfiltering, dmintro, dmopt, dmregions, dmsyntax

gui
: analysis−menu, gui

modules
: paramio, pixlib, stackio, varmm

sherpa
: sherpa_eval

slang
: math, overview, slang, tips, variables

tools
: ascii2fits
