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

AHELP for CIAO 3.4

configure

Context: concept

Jump to: Description Examples CHANGES IN CIAO 3.0 See Also

Synopsis

Configuration and customization of CIAO

Description

There are a number of ways that CIAO can be configured and customized. This document highlights these methods and points you to the relevant documentation.

Resource files

Each of Sherpa, ChIPS, and the Varmm library can be customized by a resource file. These files are evaluated by the respective program/module whenever they are started and so provide an easy way of making global changes to the system set up (you do not need these files in order for the programs to run).

ChIPS

The resource file for ChIPS is searched for (in order of preference): $CHIPSRC, $PWD/.chipsrc, and $HOME/.chipsrc.

This file can contain any valid ChIPS command, with blank lines and lines beginning with "#" being ignored. S−Lang statements can also be included: the same rules apply as at the ChIPS prompt, so variables do not need to be declared and statements do not need to end in a semi−colon but each S−Lang statement must fit onto one line.

The contents of this file are evaluated whenever a copy of ChIPS is started. This includes when Sherpa is started, or when the first plot is created from Prism, or when a S−lang script calls 'require("chips");' [or 'require("sherpa");'], as well as when ChIPS itself is started.

Sherpa

Sherpa has two resource files, one intended for general options and one for commands that change the Sherpa configuration (or "state") object. The general resource file is searched for (in order of preference): $SHERPARC, $PWD/.sherparc, and $HOME/.sherparc. The configuration−specific resource file is searched for (in order of preference): $SHERPA_STATE_RC, $PWD/.sherpa−state−rc, and $HOME/.sherpa−state−rc. Both files can exist: the .sherparc file is loaded first and then the .sherpa−stare−rc file.

These files can contain any valid Sherpa command, with blank lines and lines beginning with "#" being ignored. Since ChIPS is automatically loaded by Sherpa then you can also include calid ChIPS commands in these files too. S−Lang statements can also be included: the same rules apply as at the Sherpa prompt, so variables do not need to be declared and statements do not need to end in a semi−colon but each S−Lang statement must fit onto one line.

Since the .sherpa−state−rc file can be automatically created by the save_state() command (see "ahelp save_state") it is suggested that you only make changes to the .sherparc file.

The contents of thes files are evaluated whenever a copy of Sherpa is started: this includes importing the Sherpa module into a S−Lang script via 'require("sherpa");'.

Varmm

The location of the Varmm configuration file is $HOME/.varmmrc.

Unlike the ChIPS and Sherpa resource files, the Varmm file must be written in pure S−Lang. This means that S−Lang statements that span multiple lines − such as an if statement − can be included but you must define all variables and semi−colons are required to indicate the end of a function.

Examples of the three files are given in the Examples section below and further details can be found in "ahelp chips", "ahelp sherpa", and "ahelp varmm".

Since these files are loaded each time an instance of the program is started, they should probably be used to set up required defaults and common commands, with less−used commands loaded as required. For example, since Sherpa loads both Varmm and ChIPS, then each time Sherpa starts the Varmm, ChIPS, and Sherpa resource files are loaded (in this order) if present.

Although the resource files discussed in the Examples section create screen output, this is for demonstrative purposes. We recommend that, in general, the resource files do not contain any command that creates either text, or graphical, output.

Configuring S−Lang

When S−Lang tries to load a file using evalfile(), or load a module using require() or import(), it looks in a set of directories. The paths can be accessed using the get_slang_load_path(), get_import_module_path(), set_slang_load_path(), set_import_module_path(), append_to_slang_load_path(), and prepend_to_slang_load_path() functions from the S−Lang Run−Time library.

The CIAO environment is set up to add CIAO−specific entries to these paths via the initalization scripts described in the "Customizing CIAO" section below. Additional paths can be added by the user by setting the

```
SLANG_SCRIPT_PATH
```

and

```
SLANG_MODULE_PATH
```

environment variables before running these scripts (or adding to the environment variables after setting up CIAO).

Configuring the look of the GUIs

User−dependent configuration of the look and behaviour of the GUIs is possible as described in "ahelp gui". It is possible to change items such as imager preference (ds9 vs saotng), dynamic vs private color maps, application foreground and background colors, and fonts.

Since the resource file contains many X/motif related entries, it is treated in a similar manner to X resource files. The default version is stored in $ASCDS_INSTALL/config/system/CXCdefaults − this file can be edited and placed in $HOME/.CXCdefaults or $XENVIRONMENT (the latter must include both the path and name of the file). Further details can be found in "ahelp gui".

Changing the Analysis Menu of the GUIs

All the GUIs except for firstlook contain a user−configurable "Analysis" menu. The default settings contain a list of all the CIAO tools, which allows users to run them from the GUIs rather than the command line (with parameter settings being done by peg, the graphical parameter editor introduced in CIAO 3.0). For CIAO 3.0 the file format controlling the menu contents has been changed to use the same syntax as the ds9 "Analysis" menu (although the old syntax is still recognised but its use is deprecated). See "ahelp analysis−menu" for a detailed description of this feature.

Customizing CIAO

The CIAO installation contains default files which, when sourced, set the necessary environment variables that allow the CIAO software to be run. The files are called $ASCDS_INSTALL/bin/ciao.*sh, with different versions depending on which UNIX shell is being used (the $ASCDS_INSTALL/README file gives our suggested method for sourcing/running the necessary file, by way of a shell alias):

- *.bash − for bash
- *.csh − for csh/tcsh
- *.ksh − for ksh
- *.sh − for sh

Since these files are simple text files, you may copy and edit them to customize your own set up. Each file contains extensive comments that describe how they work, and how to change them.

The $ASCDS_INSTALL/bin/ciao.*sh files can also be changed by re−configuring the CIAO installation; for further details see step 4 of the "Binary Distribution Installation Instructions" section (or step 3b of the "Compiling Instructions" section) of $ASCDS_INSTALL/README for more information.

Example 1

The following example, if saved as $HOME/.chipsrc, will define a S−Lang command q() that mimics the ChIPS quit command. Since the command takes no arguments, the brackets need not be included when calling the function (ie this is for people who just want to type q to end ChIPS). Several settings of the ChIPS state object (see "ahelp chips") are also changed to provide a different set of plot defaults (curves are plotted as red lines). Finally a message is printed to STDOUT to let you know the file has been processed.

```
# Example $HOME/.chipsrc file
define q() { () = chips_eval("quit"); }
chips.symbolstyle = _chips−>none
chips.curvestyle = _chips−>simpleline
chips.curvecolor = _chips−>red
print("$HOME/.chipsrc has been executed")
```

Since Sherpa starts ChIPS, this file will also be evaluated whenever Sherpa is started. In this case the "q" command can also be used to exit the Sherpa session. Similarly, any plot created by prism will be done using the plot attributes defined in this file.

Example 2

Storing the following code in $HOME/.sherparc will set the default values of Sherpa's fitting statistic to be "CHI DVAR", fitting method to be "LEVENBERG−MARQUARDT", and then ensure that the fit will be refined using the SIMPLEX algorithm.

```
# Example $HOME/.sherparc file
statistic chi dvar
method levenberg−marquardt
levenberg−marquardt.smplx=1
```

Example 3

Unlike the Sherpa and ChIPS resource files, the Varmm resource file must contain valid S−Lang code. The example below, if saved to $HOME/.varmmrc, changes the value of one of the settings of the Varmm state object (see "ahelp varmm" for more information on this), and defines a function − compute_average() − that can be used by ChIPS and Sherpa sessions just like any other Varmm function (the function is intended to be demonstrative rather than particularly useful).

```
% Example $HOME/.varmmrc file
%
% change a setting of the Varmm state object
varmm.caseinsen = 1;
%
% return the average of two numbers/arrays
define compute_average (x, y)
{
    variable s = x + y;
    return s / 2.0;
}
```

The following shows part of a Sherpa session started after the Varmm resource file was created.

```
sherpa> print(compute_average(2,4))
3
sherpa> print(compute_average([2,3,5],[5,3,2]))
3.5
3
3.5
```

CHANGES IN CIAO 3.0

Sherpa configuration files

Sherpa now follows ChIPS' behaviour in looking for its configuration file in $HERPARC, $PWD/.sherparc, and then $HOME/.sherparc. In CIAO 2.3 Sherpa only looked for $HOME/.sherparc.

Sherpa also uses a .sherpa−state−rc file (with the same locations as above). However, this is primarily used by the "save_state()" routine and so care should be taken if you edit this file.

Analysis menus

The system used to configure the contents of the Analysis menu displayed in the CIAO GUIs has been changed to use the same format as used in ds9. See "ahelp analysis−menu" for more information. The old format is still understood but its use is deprecated.

See Also

chips
: chips

gui
: analysis−menu, gui

modules
: varmm

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