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

AHELP for CIAO 3.4

xpa

Context: modules

Jump to: Description Example An example of how to use the xpa module CHANGES IN CIAO 3.2 See Also

Synopsis

The S−Lang interface to the XPA library.

Description

The xpa module is the interface between the S−Lang interpreter (see "ahelp slang") and the XPA library. This document provides an overview of the features of the xpa module available in S−lang, and tips for using it efficiently in a S−Lang program. Detailed descriptions of each function are provided by individual ahelp pages.

The xpa module is not available by default; to use it in a S−Lang program, it must be loaded using:

```
require("xpa");
```

Functions provided by the module

The following functions are provided by the module; use "ahelp <function>" to get a detailed description of a function:

High−Level Interface

- xpaset
- xpaget
- xpaaccess

Low−Level Interface

- XPAOpen
- XPAClose
- XPAGet
- XPAGetB
- XPAGetToFile
- XPASet

Using built−in xpa variables and types

The xpa module defines a number of intrinsic variables, such as slxpa_errno, _slxpa_version, xpa_version, and XPA_MAXHOSTS as well as a new type, XPA_Type. The variables are described in their own ahelp pages. The new type, XPA_Type, is analogous to a unix file handle, and is used for high−volume get and set calls to avoid recreating a connection each time.

Controlling an application via XPA

The commands that can be sent to a particular XPA access point are application specific. Each particular program − such as ds9 or prism − has its own set of commands. For CIAO programs see both the ahelp page of the application and "ahelp session" for further information. The XPA Access Points section of the ds9 documentation lists the commands understood by ds9.

XPA calls versus XPA−aware

Importing the xpa module allows use of xpa calls to get or set information in xpa−aware programs, such as ds9, prism, or chips. However, this does not work in reverse; sherpa with xpa imported does not become xpa−aware.

Example

```
chips> require("xpa")
chips> system("ds9 &")
chips> xpaset("ds9","file acisf03662N001_evt2.fits")
1
chips> print( xpaget("ds9","file") )
acisf03662N001_evt2.fits[EVENTS]
chips> xpaset("ds9","frame new")
1
chips> array = Short_Type[256,256]
chips> for (i=0;i<256;i++) for (j=0;j<256;j++) array[i,j] = 16*i+j
chips> xpaset("ds9","array [dim=256,bitpix=16]",array)
1
chips> files = xpaget("prism","file")
chips> cols = xpaget("prism","selected cols")
chips> cols = strtrans(cols[0]," ",",")
chips> chips_eval(sprintf("curve \"%s[cols %s]\"",files[0],cols))
```

An example of how to use the xpa module

The following routine can be used to send an array of data to ds9 automatically:

```
  %
  % Usage:
  %  send_image( pixels );
  %
  require("xpa");
  define send_image(pixels) {
    variable dims, dim, datatype, cmd;
    (dims, dim, datatype) = array_info(pixels);

    % The following is DS9 specific;  fits_bitpix is part of CIAO
    cmd = sprintf("array [xdim=%S,ydim=%S,bitpix=%S]",dims[0],dims[1],
                                          fits_bitpix(datatype));
    () = xpaset("ds9",cmd,pixels);
  } % define send_pixels()
```

With the above function, one can send any 2 dimensional array to ds9 without extracting the X and Y dimensions or the size of the data elements.

```
chips> evalfile("send_image.sl");
1
chips> im = readfile("acis_evt2.fits[bin sky=8]")
chips> send_image(im.pixels)
chips> % Modify the data in an unusual fashion and display in a new frame
chips> xpaset("ds9","frame new")
1
chips> im.pixels[where(im.pixels > 10)] = 10
chips> send_image(im.pixels)
```

CHANGES IN CIAO 3.2

Loading the module

The module can now be loaded by using the

```
require("xpa");
```

statement, although the previous method (loading with the import command) still works.

Backwards Compatability

The default behavior of the xpaset(), xpaget(), XPAGet(), XPAGetToFile(), and XPASet() functions has changed since CIAO 3.1. The old behavior can be retained by declaring the variable

```
_CIAO3_XPA_COMPAT_
```

in the Global namespace prior to the first loading of the module. As an example, after:

```
if (0 != _featurep("xpa"))
  error("The XPA module has already been loaded");
public variable _CIAO3_XPA_COMPAT_;
require("xpa");
```

then the CIAO 3.1 versions of the commands will be used. The first statement, fearuting the _featurep() function, is not necessary, but added as a precaution to check that the XPA module has not previously been loaded.

Use in Sherpa

Please note that Sherpa loads the XPA module with the _CIAO3_XPA_COMPAT variable set, so any use of the module from Sherpa will use the compatability mode.

Variable changes

The version number of the module is now stored in the

```
_slxpa_version
```

variable, rather than the

```
slxpa_version
```

variable.

New functionality

The XPAGetB() routine has been added to the module. This routine provides access to data from XPA servers as binary, rather than string, data.

Users of ds9 may wish to use the ds9 module for interacting with ds9 via XPA. See "ahelp modules ds9" for more information.

See Also

xpa

_slxpa_version, slxpa_errno, xpa_maxhosts, xpa_version, xpaaccess, xpaclose, xpaget, xpagetb, xpagettofile, xpaopen, xpaset

---

The Chandra X−Ray Center (CXC) is operated for NASA by the Smithsonian Astrophysical Observatory.
60 Garden Street, Cambridge, MA 02138 USA.
Smithsonian Institution, Copyright © 1998−2006. All rights reserved.

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