Reference Manual for SLxpa Version 0.5.2

Michael S. Noble, mnoble@space.mit.edu Jul 31, 2010

Preface

SLxpa is a software package which provides XPA bindings for the S-Lang scripting language. It was originally developed in 2002 at the Smithsonian Astrophysical Observatory, as part of the CIAO analysis package developed to support the Chandra X-Ray Observatory, and is covered by the notice COPYRIGHT.SAO in the root of the CIAO distribution (and included below).

The standalone module (distributable independently of CIAO) was created at the MIT Center for Space Research between 2003 and 2004.

Contents

Chapter 1

Introduction

SLxpa binds the XPA library to the S-Lang language. It provides an importable module and associated .sl script which make XPA callable directly from S-Lang scripts. The S-Lang interpreter is part of the widely-used, open-source library of the same name, and provides a scripting environment well-suited for scientific and engineering tasks, due to the powerful, compact, fast, and robust multidimensional numerical capabilities that are native to the language.

XPA is a set of libraries (ANSI C, Tcl/Tk) and commandline tools, written at the Smithsonian Astrophysical Observatory, which provide a clean and simple IPC (inter-process communication) mechanism that is also surprisingly powerful. It can be used to build loosely-coupled client/server programs that interact either on a single machine or across local and wide-area networks; we have also found it useful for automated regression testing, especially for GUIs that are traditionally difficult to test in this manner. The XPA library provides much of the functionality of more well-known mechanisms like CORBA, Java RMI, and the now-defunct ToolTalk, but is much smaller and, we feel, can be easier to learn, embed, deploy, and utilize.

The `SAOds9` image display and analysis tool is perhaps the most well-known XPA command server. Also written at the Smithsonian Astrophysical Observatory, SAOds9 is one of the single most widely used programs in observational astronomy. `CIAO`, a software system created to assist in the analysis of data generated by the Chandra X-Ray Telescope, provides a number of programs (e.g. `ChIPS` and `Prism`) that function both as XPA clients (by interacting with DS9 and each other) and servers (by providing XPA access points of their own). Other software employing XPA includes `fv` and `POW` (a FITS file viewer and plotting tool, respectively, available within both the `FTOOLS` and `HEASOFT` astronomy software suites), as well as the `SAS` package created for the XMM-Newton X-Ray Telescope. A set of Perl bindings to XPA also exists in CPAN.

1.1 Brief History

S-Lang bindings for XPA were originally developed at the Smithsonian Astrophysical Observatory in 2002, as part of `CIAO`, an analysis software package developed in part to support the Chandra X-Ray Telescope. The SLxpa package, created at the MIT Center for Space Research between 2003 and 2004, adds significant enhancements to the original bindings, in order to:

- take better advantage of S-Lang features (such as returning multiple values from a function)
- clarify precisely what values slxpa_errno may hold
- provide new get routines which return `BString_Type` data
- indicate the number of servers contacted by `xpaset`
- return *names* and *messages* arrays from `XPAGet` and `XPASet`
- properly flush files created during `XPAGetToFile` calls
- simplify runtime debugging with *slirp_debug_pause* (see documentation at *the SLIRP website* http://space.mit.edu/~mnoble/slirp/)
- improve internal robustness
- provide CIAO-independent example and test scripts
- provide improved documentation

Moreover, SLxpa extends their scope of potential use to a considerably wider audience, in that the package may be obtained independently of CIAO, allowing developers to avoid downloading hundreds of megabytes of astronomical software and data just to obtain a 100 Kb module.

1.2 Backwards Compatibility

As noted in the respective function descriptions, some of the SLxpa enhancements described above change the semantics of the XPA bindings as originally developed within CIAO. However, SLxpa may still be used as a drop-in replacement for those bindings, simply by declaring the variable `_CIAO3_XPA_COMPAT_` in the `Global` namespace prior to loading the package, e.g. as

```
public variable _CIAO3_XPA_COMPAT_;
```

The variable need not be initialized to any particular value.

1.3 Using the XPA Module

1.3.1 Building

SLxpa ships with a configure script generated by *autoconf*, and in most cases can be built by issuing the following standard commands within a UNIX-like (e.g. Linux or Cygwin) environment:

```
./configure [options]
make
make install
```

The third step is optional, but recommended. Version 2.1.4 or later of XPA is recommended. Versions of SLxpa prior to 0.5 were compatible with S-Lang 1, but SLxpa versions 0.5 and later require S-Lang 2.1 or later (e.g. to support the use of named qualifers in function calls). The `./configure -help` command will display all available configure options.

1.3.2 Running

To use the `XPA` module in a S-Lang script it is first necessary to make the functions in the package known to the interpreter, via

```
() = evalfile ("xpa");
```

or, if the application embedding the interpreter supports the `require` function,

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

may be used. You may also load the package into a namespace, such as

```
() = evalfile ("xpa", "xpans");
```

This would place the XPA symbols into the `xpans` namespace (creating it, if necessary). Once the package has been loaded functions and variables defined within may be used in the usual way, e.g.

```
require ("xpa");
.
.
() = xpaset("ds9","quit");
```

Finally, recall that the S-Lang `autoload` function may be used to avoid explicitly `evalfile`-ing or `require`-ing the package prior to using the functions defined within.

The remainder of this document assumes the reader is familiar with the XPA suite of command line tools and/or the XPA application programming interface (api). For more information consult the *XPA website* http://hea-www.harvard.edu/RD/xpa .

1.4 A Word About Examples

To make the most of the examples given below it is encouraged that you run them and experiment. In support of that you should first:

- ensure that XPA and DS9 (version 3.0 or later) are installed on your system
- build SLxpa (installation is not required to run the examples)
- change to the `examples` directory of the SLxpa distribution
- invoke a copy of DS9

The examples reference programs mentioned in the introduction with which you may not be familiar. Do not fret, as you need not actually run an example within the given application for it to still serve most of its didactic purpose, because what's being shown are not specific features of the application, per se, but rather how XPA function calls may be made *through the S-Lang interpreter*. The XPA functions may be called in the exact same manner from any application which embeds the S-Lang interpreter and supports module importation. In fact, most of the examples can be cut and pasted into a S-Lang script and then invoked from within the S-Lang shell (`slsh`, distributed with the S-Lang library, and assumed to be installed on your system).

1.5 Interactive Image Analysis with SAOds9

As described in chapter 5 (DS9_Package), SLxpa also bundles a script which streamlines interactions with the SAOds9 image display and analysis tool.

By augmenting the numerical strength and modular extensibility of the S-Lang platform with the imaging capabilities of SAOds9, the DS9 script can make a powerful addition to your image analysis toolset.

Chapter 2

Module Overview

The SLxpa api provides both high-level and low-level views on the underlying XPA library. At the high-level SLxpa seeks to mimic the three most commonly used XPA command line utilities: `xpaset`, `xpaget`, and `xpaaccess`. It is anticipated that the corresponding S-Lang functions will also be the most commonly used SLxpa features.

At the low-level SLxpa aims at mirroring the XPA api more closely. To that end, rather than replicate the XPA documentation here we simply note discrepancies between the respective apis and refer the reader to the *XPA website* .

Note that for brevity we use the terms *XPA access point* and *XPA command*, and likewise *XPA server* and *target*, interchangeably.

2.1 Versioning

SLxpa indicates version information in the variables

`_xpa_module_version`	`An integer containing the value (10000*major_ver) + (100*minor_ver) + micro_ver`
`_xpa_module_version_string`	`A string containing the value major_ver.minor_ver.micro_ver`

Earlier releases of SLxpa used the variables `_slxpa_version` and `_slxpa_version_string` to convey this information, but these have been deprecated. As a user convenience SLxpa also indicates the version of XPA against which it was built, via the floating point variable

```
xpa_version
```

2.2 Omitted Arguments

If a function argument is omitted at invocation time, and a default value is given in the functions *Usage* statement, the underlying XPA C function will be invoked as if the default value were specified

for the omitted argument. Note that, as is the case with, say, C++, in order to default the *Nth* argument each of arguments *1* through *N-1* must be specified.

2.3 Error Checking

SLxpa introduces the **slxpa_errno** variable, which indicates the last error which occurred internal to the XPA module. The module zeros **slxpa_errno** at import time, and explicitly assigns only the following values

```
1               XPA_COULD_NOT_CONNECT
2               XPA_SERVER_ERROR
3               XPA_MALLOC_ERROR
```

during subsequent execution. As with the **errno** variable defined by C, it is the caller's responsibility to zero **slxpa_errno** when needed.

2.4 Contacting Multiple Servers

The `XPA_MAXHOSTS` environment variable, when set, signifies the upper limit for the number of application servers that any single SLxpa call will contact. By default this value is 5. Note that this variable is also utilized by the XPA library itself, as are others described in the XPA documentation.

2.5 Unimplemented Functions

2.5.1 Server

The XPA module supports only the client portion of the XPA library; no server functions have yet been wrapped.

2.5.2 Client

Several client functions have been omitted:

- `XPAInfo`
- `XPANSLookup`
- `XPASetFd`

It is unclear if any applications have been written which provide *info* XPA access points. Although technically speaking SLxpa does provide a wrapper for `XPASetFd`, namely `XPASetFromFile`, at present it is only an empty stub.

Chapter 3

High Level XPA Interface

3.1 xpaset

Synopsis

Send data to one or more XPA servers

Usage

```
Integer_Type     xpaset(targets, xpa_cmd [, param, param, ... [,data]])
```

Description

This function is used to send information or commands to one or more XPA server(s). The return value indicates the number of application servers contacted.

Example

```
() = xpaset("ds9","file new stars.fits")
() = xpaset("ds9","cmap","heat")
```

These commands load a new file into DS9 and select the `heat` colormap, discarding the return values. They also demonstrate the flexibility with which XPA parameters may be passed to the server, either by concatenation with the XPA command name or as distinct comma-separated arguments.

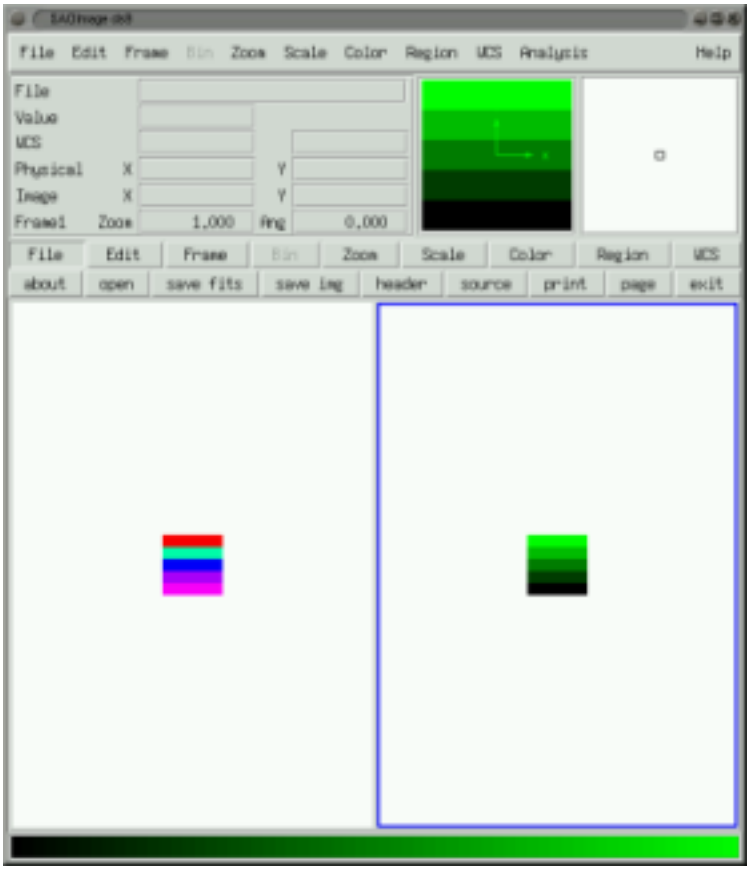
Internally the function automatically appends all scalar parameters (char, short, integer, long, float, or string) to the XPA command name before transmission. When a non-scalar parameter argument is present (e.g. an array of floats) it will be treated as *data* (see the XPA documentation) for the command, and all subsequent arguments will be ignored.

The next example creates two test patterns directly from an in-memory S-Lang array:

```
() = evalfile("./setup.sl");

variable arr = Short_Type[2500];
arr[[0:499]] = 100;
arr[[500:999]] = 200;
```

```
arr[[1000:1499]] = 400;
arr[[1500:1999]] = 800;
arr[[2000:2499]] = 1600;

() = xpaset("ds9","frame new");
() = xpaset("ds9","cmap rainbow");
() = xpaset("ds9","array [dim=50,bitpix=16]",arr);

() = xpaset("ds9","frame new");
() = xpaset("ds9","cmap green");
() = xpaset("ds9","array [dim=50,bitpix=-64]",log10(arr));
```

and should yield a result which looks like



The first block of `xpaset` invocations sends 2500 short integers, as binary data, to the DS9 *array* command, along with instructions that it be visualized in the rainbow colormap and treated as a 50x50 image with 16 bits per pixel.

The subsequent statements generate a new image by performing a base 10 logarithmic transform of the original image, indicating how simple and natural it can be to perform sophisticated image manipulations. Moreover, instead of the `log10` function one could in principle substitute any numerical function callable from S-Lang scope, such as that which might be provided

by the *GSL* module (which binds the GNU Scientific Library to S-Lang), or even your own homegrown C or FORTRAN codes. The entire script may be executed from the examples directory, under UNIX for instance, via

```
your_unix_machine % slsh testpattern.sl
```

See Also

`XPASet, xpaget`

Compatibility Note

In the CIAO 3.0 bindings this function did not return a value. This was intended to reflect the most common usage, namely that the the *number of servers contacted* is usually ignored at the command line, avoiding superfluous

```
() = xpaset(...);
```

notation in scripts (which would be required in S-Lang 1.x to explicitly pop the return value off the stack), in favor of the "more natural"

```
xpaset(...);
```

The caller could determine the number of servers contacted during these calls by inspecting the `slxpa_errno` variable upon return. See section 1.2 (Backwards_Compatibility) to restore this behavior.

3.2 xpaget

Synopsis

Retrieve information from XPA servers, by name

Usage

```
Array_Type     xpaget(targets [, xpa_cmd = ""])
```

Description

This function is used to retrieve information from the specified XPA server(s). Results are returned as an array of strings, one from each application server contacted.

When no XPA command is given the application(s) contacted typically return a list of the XPA commands that the application supports.

Example

```
vmessage( xpaget("ds9") [0] );
```

This command dumps to the screen the entire list of XPA commands supported by DS9. As another example, if DS9 were invoked as

```
your_unix_machine % ds9 examples/stars.fits &
```

then the following

```
vmessage( xpaget("ds9","file") [0] );
```

would print

```
examples/stars.fits
```

See Also

`XPAGet, XPAGetToFile, xpaset`

Compatibility Note

In the CIAO 3.0 bindings this function would return a single string, rather than an array containing only 1 string, when only a single application server responded. See section 1.2 (Backwards_Compatibility) to restore this behavior.

3.3 xpaaccess

Synopsis

Determine how many XPA servers offer the specified command(s)

Usage

```
Integer_Type     xpaaccess(targets [, acc_mode = "gs"])
```

Description

Returns 0, 1, or more to indicate how many XPA servers are running which offer commands (access points) matching the given target template.

Example

Bring down any running instances of DS9, then launch it 3 times via

```
your_unix_machine % ds9 &
```

and observe how

```
your_unix_machine % slsh <<EOT
import("xpa");
message(string(xpaaccess("ds9")));
EOT
```

returns the value 3. Similarly, if we then did

```
your_unix_machine % ds9 -title BOB &
```

then issuing the above `xpaaccess` command again would still indicate that only 3 instances of DS9 were running, while

```
xpaaccess("BOB");
```

would return 1.

See Also

`The xpaaccess command line utility.`

Chapter 4

Low Level XPA Interface

Note that in the SLxpa api the *mode* parameter has been moved to the end of the argument list for those functions which utilize such. This simplifies scriptwriting, since clients rarely utilize a *mode*, by allowing the parameter to be omitted from most calls.

4.1 XPA_Type

SLxpa introduces the `XPA_Type` S-Lang type to wrap the `XPA` structure defined within the XPA C library. Instantiated variables of this type are opaque, meaning their contents cannot be inspected in S-Lang scope.

4.2 XPAOpen

Synopsis

Allocate a persistent client handle

Usage

```
XPA_Type     XPAOpen([mode])
```

Description

Explicitly instantiate a persistent XPA_Type client handle for use in subsequent XPA calls. This is most useful for high-volume traffic, to avoid the cost of setting up and tearing down the communications channel to the server(s) during each message exchange. The mode parameter is currently ignored by XPA, and so may be omitted from SLxpa calls with no consequence.

See Also

`XPAClose`

4.3 XPAClose

Synopsis

Close a persistent client handle

Usage

```
Void_Type     XPAClose(XPA_Type xpa)
```

Description

Explicitly close an XPA handle and free its resources. Strictly speaking it is not necessary to call this function, since SLxpa automatically closes XPA connections when they go out of scope. This routine is provided mainly for completeness.

See Also

`XPAOpen`

4.4 XPAGet

Synopsis

Retrieve information from XPA servers, using client handle, as strings

Usage

```
(results, names, messages)  =  XPAGet(handle, targets, xpa_cmd
                               [,max_recipients = XPA_MAXHOSTS [, mode = ""]])
```

Description

Similar to `xpaget`, but offers finer control and returns additional information. The `results`,`names`, and `messages` return values are all arrays of `String_Type`, whose length indicates the number of servers contacted.

In practice most XPA commands do not return binary data, but rather a small number of discrete datapoints (e.g coordinate values, filename(s), upper/lower limits). This routine is well suited for retrieving results from such access points. Use either `XPAGetB` or `XPAGetToFile` to retrieve binary data.

See Also

`xpaget, XPAGetB, xpaset, XPASet`

Compatibility Note

In the CIAO 3.0 bindings this function , like `xpaget`, would return a single string, rather than an array containing only 1 string, when only a single application server responded. Moreover, the return value did not reflect the *lens*, *names*, or *messages* arrays populated by underlying C function. See section 1.2 (Backwards_Compatibility) to restore this behavior.

4.5 XPAGetB

Synopsis

Retrieve information from XPA servers, using client handle, as binary data

Usage

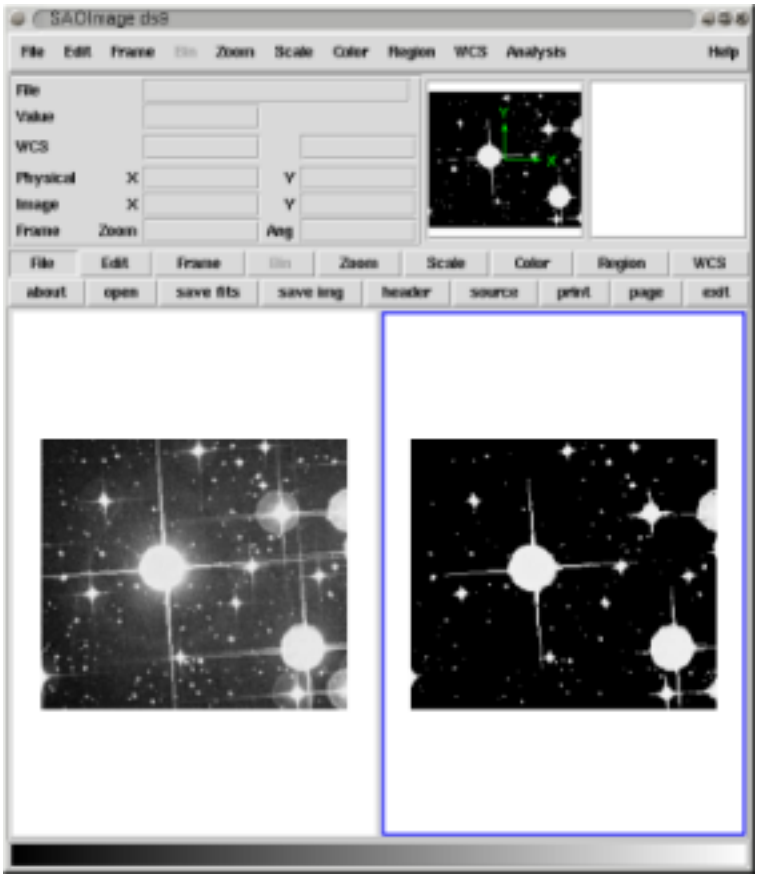
```
(results, names, messages)  =  XPAGetB(handle, targets, xpa_cmd
                               [,max_recipients = XPA_MAXHOSTS [, mode = ""]])
```

Description

Similar to `XPAGet`, the only difference being that the results array is of type `BString_Type`.

Example

Consider the visual



generated by the following script:

```
() = evalfile("./setup.sl");

() = xpaset("ds9","file new stars.fits");

% Retrieve the file as raw binary data
```

```
variable result;
(result, , ) = XPAGetB( XPAOpen(), "ds9", "array");

% Zero pixels with less than 200 counts in brightness
variable image = bstring_to_array(result[0]);
image [ where (image < 200) ] = 0;

% Display the zeroed image.  Note that newer versions of DS9
% provide a way of retrieving the dimensions of displayed images,
% but for convenience we simply hardcode 255x225 here
() = xpaset("ds9","frame new");
() = xpaset("ds9","array [xdim=255,ydim=225,bitpix=8]",image);

% Re-retrieve it, to test the interface
(result, , ) = XPAGetB( XPAOpen(), "ds9", "array");
variable test = bstring_to_array(result[0]);
if (length (where (test != image)))
   () = fprintf (stderr, "XPAGetB has failed\n");
```

Here we've combined `XPAGetB` with the S-Lang binary string manipulators, yielding a powerful mechanism for transmitting DS9 images between applications while using no intermediate files.

See Also

`xpaget, XPAGet, xpaset, XPASet`

4.6 XPAGetToFile

Synopsis

Retrieve information from XPA servers, using client handle, write results to file

Usage

```
(names, messages)  =  XPAGetToFile(handle, targets, xpa_cmd
                                                [, fileid = stdout
                                                [, max_recipients = XPA_MAXHOSTS
                                                [, mode = ""]]])
```

Description

This is the SLxpa wrapper for `XPAGetFd`, renamed slightly to connote increased generality. It is similar to `XPAGet` and `XPAGetB`, except that results are written (as binary data) to the given file identifier, instead of being returned on the stack.

The file identifier may one of

- the name of a file that will be written to (and created or overwritten as necessary)
- a file pointer, such as returned by `fopen()` or `popen()`
- a file descriptor, such as returned by `open()` or `dup_fd()`

Unlike the C api, at present only 1 unique file may be specified in the S-Lang layer.

Example

Rather than using binary string data the previous example could be rewritten to make use of the UNIX *mmap* function as wrapped by the *varray* module bundled with the S-Lang distribution:

```
 () = evalfile("./setup.sl");

import("varray");                              % module bundled with S-Lang

% Newer versions of DS9 provide a way of retrieving the dimensions of
% displayed images, but for convenience we simply hardcode them here
variable names, messages, size = 255*225;
variable handle = XPAOpen();

(names,messages) = XPASet(handle, "ds9", "file new stars.fits");
check_errors(names,messages);

variable tmpfile = "tempfile.dat";
() = remove(tmpfile);

(names,messages) = XPAGetToFile( handle ,"ds9", "array", tmpfile);
check_errors(names,messages);

variable finfo = stat_file(tmpfile);
if (orelse {finfo == NULL}  {finfo.st_size != size})
   error(tmpfile +" for mmap() is missing or of incorrect size");

variable arr = mmap_array(tmpfile,0,UChar_Type,size);

% duplicate the mmap'd array, so we can modify its content
arr = @arr;
arr [ where (arr < 200) ] = 0;

() = xpaset("ds9","frame new");
() = xpaset("ds9","array [xdim=255,ydim=225,bitpix=8]",arr);

() = remove(tmpfile);
```

See Also

`XPAGet, xpaget, xpaset`

4.7 XPASet

Synopsis

Send data to one or more XPA servers, using client handle

Usage

```
(names, messages)  =  XPASet(handle, targets, xpa_cmd
                      [, max_recipients = XPA_MAXHOSTS
                      [, data = NULL
                      [, mode = ""]]])
```

Description

Similar to xpaset(), but with more user control. The `names` and `messages` return values are, again, arrays of `String_Type`, whose lengths indicate the number of servers contacted.

Note that the *len* parameter present in the C function signature (which indicates the number of data bytes to be transmitted) is not required in S-Lang, since that can be determined automatically.

See Also

`xpaset, xpaget, XPAGet`

Compatibility Note

In the CIAO 3.0 bindings this function did not return the *names* or *messages* arrays populated by underlying C function, but rather returned only an integer indicating the number of servers contacted. See section 1.2 (Backwards_Compatibility) to restore this behavior.

Chapter 5

Using DS9 from S-Lang

This chapter describes the DS9 package, which provides a series of S-Lang functions, layered above the XPA module, that streamline interactions with the SAOds9 image display and analysis tool widely used in astronomy. We assume some familiarity with the FITS file format and related conventions, as described at the *FITS website* http://fits.gsfc.nasa.gov . The package may be accessed from any S-Lang-enabled interpreter through the usual means, such as `autoload()`, or

```
() = evalfile ("ds9");
```

or, if the application embedding the interpreter supports the `require` function,

```
require ("ds9");
```

5.1 Examples

In this section we illustrate a number of ways in which the DS9 package may be employed, with an emphasis on the common case and ease of use. For context we show the functions as they might be invoked from the interactive prompts of *ChIPS* http://cxc.harvard.edu/ciao and *ISIS* http://space.mit.edu/CXC/isis (and in the latter case assume `Isis_Append_Semicolon` = 1). The examples may also be executed in batch within `slsh` scripts, or interactively from the prompt of the *readline*-enabled `slsh`, and so forth.

Additional details are available in the function reference in the next chapter, or in the short usage message that most of the DS9 functions will emit when invoked with zero arguments. The reader may also wish to inspect the DS9-related scripts, within the *tests* subdirectory, for supplemental examples.

5.1.1 Getting Started

If DS9 is not already running it may be launched from S-Lang via commands such as

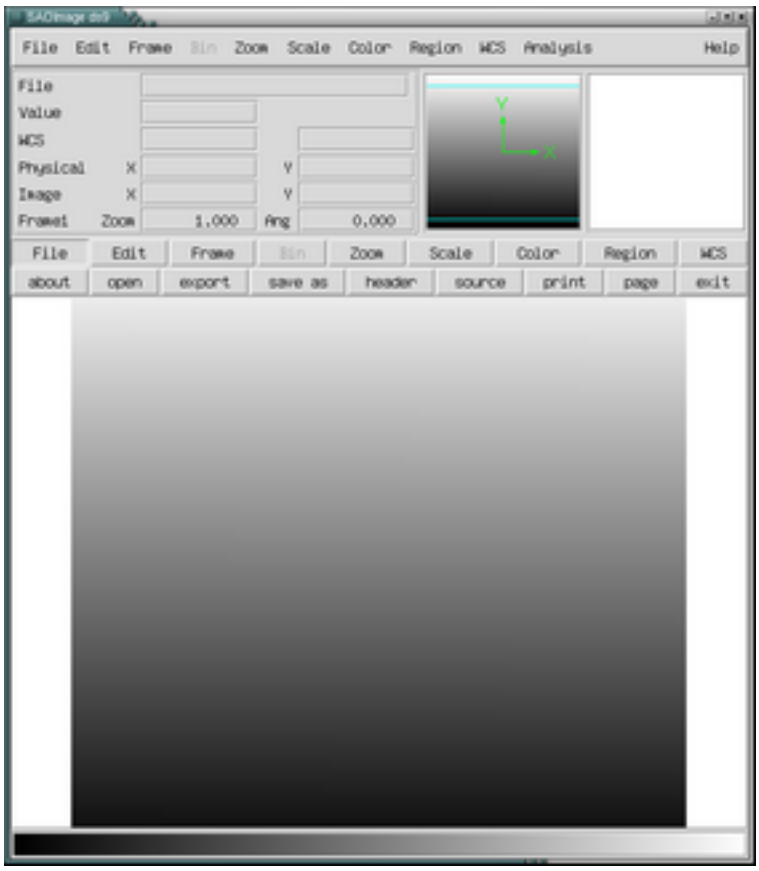
```
isis> ds9_launch
Struct_Type
```

```
isis> ds9_view( "image.fits" )

isis> ds9_view( [ 1 : 512 * 512] )
```

A second DS9 process will not be invoked here if one is already running, even if it was started outside of S-Lang. Moreover, DS9 will continue running after the S-Lang application from which it may have been launched has been terminated.

In the first command parentheses have been intentionally omitted, since `ds9_launch` may be called with zero or more arguments. Likewise, the return value, a so-called *handle* structure, has been purposely left on the stack (and is subsequently cleared/echoed by ISIS).

The second and third forms are likely to be preferred, since `ds9_view` not only launches DS9 but also causes it to immediately display the specified image. The first view command shown here simply sends the name of a file, from which DS9 will load the default image. The second is more interesting: it creates an 1D inlined array of 262144 elements, initialized to the values 1 through 512*512, and transmits the result directly to DS9, resulting in a visual which should resemble



5.1.2 Sending Images to DS9

Our next example reveals one of the most powerful features in the DS9 package, namely the `ds9_put_array` function. With it we can rewrite the preceding example as

```
isis> ds9_launch
isis> ds9_put_array( [ 1 : 512 * 512] )
```

The `ds9_put_array` method is faster and cleaner than alternatives, such as dumping data to temporary files and explicitly constructing command strings to pass to the `system()` function for launching external processes. Instead, by fostering direct communication between S-Lang and DS9 we avoid creating file litter which must be cleaned up later, as well as the need to load additional software to read and write FITS files. As a result the analysis process becomes more fluid and nimble, promoting iterative exploration.

Similarly, if DS9 is already running then

```
isis> ds9_put_file ( "image.fits" )
```

may be used to transmit the name of a file from which DS9 should visualize an image.

5.1.3 Retrieving Images from DS9

The `put` functions described above have natural inverses. For example, to retrieve the name of the file currently being displayed we might do

```
chips> filename = ds9_get_file()
chips> print(filename)
image.fits
```

or, more succinctly,

```
chips> ds9_get_file
image.fits
```

Likewise, to retrieve the actual image being displayed one might do

```
chips> image = ds9_get_array
```

following which one can perform all of the expected S-Lang array manipulations, such as

```
chips> image
UChar_Type[900,900]
```

which reveals the image dimensions, or

```
chips> image2 = sqrt(image)
```

which constructs a new image (of `Double_Type`) as the `square root` of the first, or

```
chips> ds9_put_array(image2)
```

which sends the result back to DS9.

5.2 Point and Click Interactivity

See the function reference (in the next chapter) and test scripts.

5.3 Regions

Here's the simplest case showing how regions may be retrieved directly to a S-Lang string array

```
isis> s = ds9_get_regions()
isis> print(s)
```

yielding a result like

```
# Region file format: DS9 version 4.0
# Filename: im1.fits
global color=green font=\"helvetica 10 normal\" select=1 highlite=1 edit=1 move=1 delete=1 include=1 f
image
circle(268,257,20)
```

The regions were returned in the "image" coordinate system because it is what was selected in the Region menu of the GUI. A different coordinate system may be selected using the named `coord_sys` qualifier

```
isis> s = ds9_get_regions( ; coord_sys="physical")
isis> print(s)
```

which might change the last two lines of the above output to something like

```
"physical"
"circle(4280.5,4104.5,320)"
```

Named qualifiers in S-Lang must appear after positional parameters, and are separated from them by the semicolon delimiter. Another way to achieve the same end would be to reset the region coordinate system prior to retrieving the regions themselves, such as

```
isis> ds9_send("regions system physical")
```

This setting persists until changed, allowing the region retrieval to again be as cleanly expressed

```
isis> s = ds9_get_regions()
```

as in the first example. Regions may also be saved to a file, e.g.

```
isis> s = ds9_get_regions("/tmp/my.reg")
isis> !cat /tmp/my.reg

# Region file format: DS9 version 4.0
# Filename: im1.fits
global color=green font="helvetica 10 normal" select=1 highlite=1 edit=1 move=1 delete=1 include=1 fi
physical
circle(4280.5,4104.5,320)
```

Observe that the regions were again returned in the "physical" coordinate system, because the GUI retains the previous selection.

5.4 World Coordinate Systems

Our examples so far have treated images only as arrays of raw pixel values. We now highlight how coordinate systems may be attached to these images, thereby rendering them of greater practical significance to the astronomer.

Formatted WCS Strings

Suppose we would like to attach a WCS transform described by the strings

```
"CRPIX1    256"
"CRVAL1    512"
"CDELT1    2"
"CTYPE1    X"
"CRPIX2    256"
"CRVAL2    512"
"CDELT2    2"
"CTYPE2    Y"
```

(which simply multiplies the coordinate values of each axis by 2) to the DS9 image generated by

```
chips> ds9_view( [ 1 : 512 * 512] )
```

If the transform was contained within the text file `image.wcs`, then the command

```
chips> ds9_put_wcs_keys("image.wcs")
```

would attach it to the current image. The same function would be used if the transform strings were instead contained within an 8-element array variable named `wcs`, only now it would be invoked as

```
chips> ds9_put_wcs_keys(wcs)
```

As the function reference indicates, the key/value pairs given here may also be formatted in accordance with the FITS 80 character card standard.

Raw/Unformatted WCS Values

Now suppose that we wanted to apply the transform not from formatted strings, but rather from raw S-Lang values. For example, we might apply the transform to the first axis of the image with

```
chips> ds9_put_wcs(256, 512, 2)
```

or to both axes of the image via something like

```
chips> crpix = [256, 256]
chips> crval = [512, 512]
chips> cdelt = [2, 2]
chips> ds9_put_wcs(crpix, crval, cdelt)
```

5.4.1 WCS Structures

Now suppose that the arrays given in the preceding example were morphed into fields of the same name within an S-Lang structure, such as

```
isis> s = struct { crpix, crval, cdelt, ctype, cunit}
isis> s.crpix = crpix
isis> s.crval = crval
isis> s.cdelt = cdelt
```

The entire structure could then be applied at once via

```
isis> ds9_put_wcs_struct(s)
```

5.4.2 Alternate WCS

Both FITS and DS9 allow multiple coordinate systems to be associated with an image. The DS9 module facilitates this by supporting an optional `alt_wcs_char` parameter in each of its WCS functions. For example,

```
isis> ds9_put_wcs_struct(s, 'a')
```

would create a so-called WCSa coordinate system. Similarly,

```
isis> crpix = [256, 256]
isis> crval = [512, 512]
isis> cdelt = [2, 2]
isis> ds9_put_wcs(crpix, crval, cdelt, , , 'p')
```

would create a WCSp coordinate system. As a convenience the DS9 module assigns the WCSp coordinate system as the physical coordinate system within the DS9 GUI, by transparently passing an additional FITS keyword `WCSNAMEP` with value `'PHYSICAL'`. Note that in the above call `ctype` and `cunit` have been omitted (positional parameters 4 and 5 are empty); their values will default to `NULL`. More information on how DS9 treats alternate WCS is available within the DS9 FAQ, which can be accessed through Help->FAQ->Coordinates within the GUI.

5.4.3 Erasing WCS

A transform may be removed simply by replacing it with an empty transform, such as

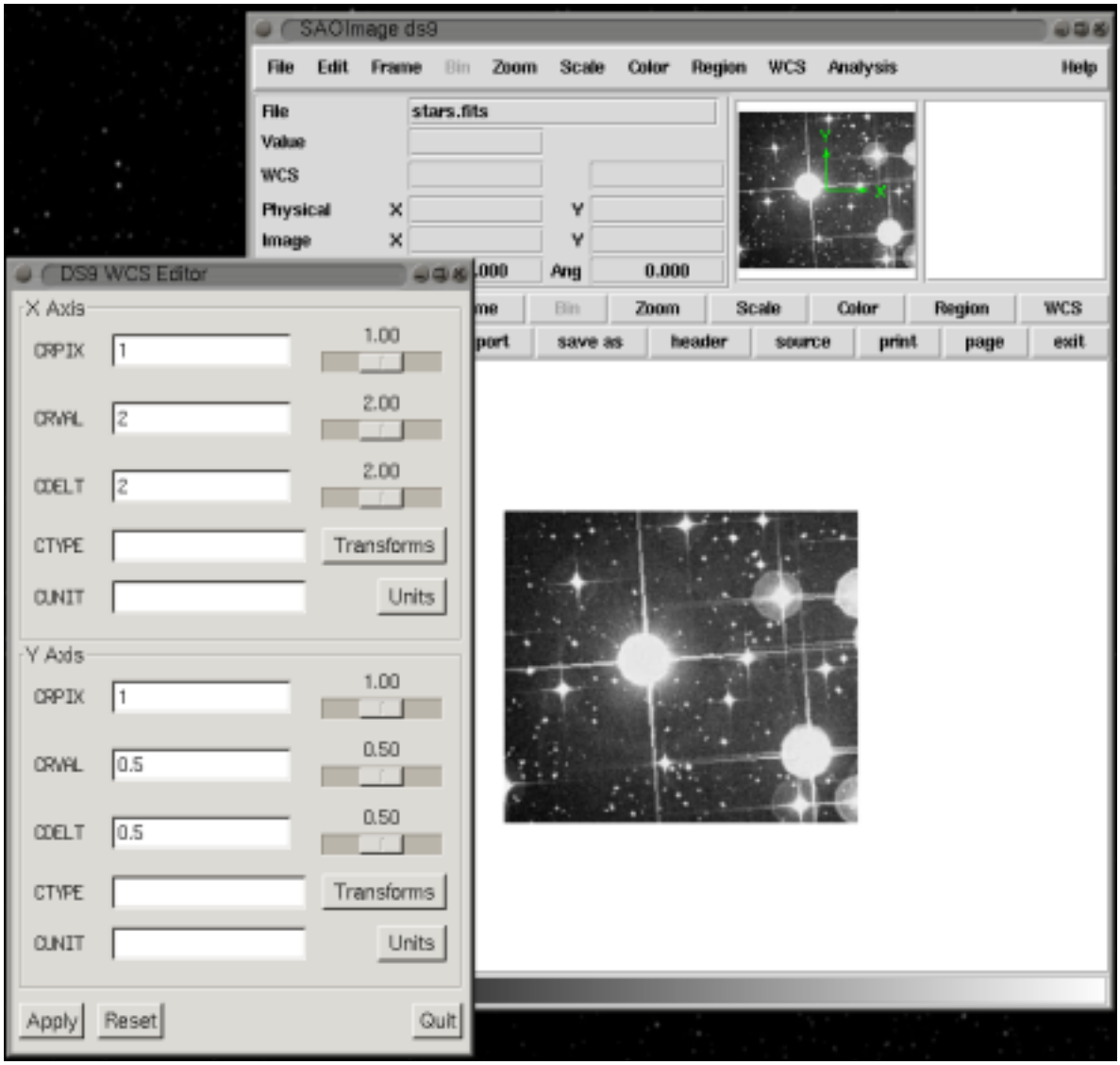
```
chips> ds9_put_wcs_keys("")
```

5.5 WCS Editor GUI

If you have *SLgtk* http://space.mit.edu/cxc/software/slang/modules/slgtk/ installed then you'll also be able to modify image WCS values from within the WCS editor guilet, launched via

```
ds9_wcs_edit
```

The next figure shows the GUI being used to double image coordinate values along the X axis, and halve them along the Y axis.



5.6 Shutting Down

A DS9 process may be terminated either interactively from the GUI or programmatically via

```
isis> ds9_quit()
```

5.7 Optional Arguments

As discussed earlier in 2.2 (Omitted_Arguments), order matters when specifying optional arguments. For example, consider `ds9_clear()`, which accepts 2 optional arguments: a handle cannot be specified when this function is invoked unless the frame argument is also specified. Fortunately, the API has been coded to accept NULL as a default value in such cases. This exploits the fact that omitted arguments delimited by commas default to the value `NULL` in S-Lang, allowing such functions to be conveniently invoked, for example, as

```
ds9_clear( , handle);
```

5.8 Return Values

Note that some routines in the DS9 package do not return a value which can be inspected for an error condition. There are two reasons for this. One is that the package is intended primarily for interactive use, so when a function fails such will be immediately evident to the user. Second, functions can only return error codes when SAOds9 itself indicates that an error condition exists, but unfortunately this information is not returned to the caller in all cases.

While XPA supports sending to and receiving from multiple servers in a single call, the functions described here return results from at most 1 instance of SAOds9.

Chapter 6

DS9 Function Reference

6.1 ds9_launch

Synopsis

Establish connection to a DS9 process

Usage

```
Struct_Type = ds9_launch( [xpa_id="ds9", ds9_cl_arg1, ds9_cl_arg2, ... [ ; timeout=num_seconds, ver
```

Description

Returns a handle to an instance of SAOds9 running with the specified XPA template identifier. If such a process is not already running then one will be launched, with the optionally command line arguments. To optimize communication with SAOds9 the returned handle should be passed to subsequent calls. The timeout= qualifier can be used to adjust how long the routine will wait to contact DS9 before giving up (the default is 30 seconds). The verbose= qualifier can be used to alter the amount of information printed to the screen while interacting with DS9.

Notes

When launched with no arguments this routine looks for/launches a default SAOds9 process, running on the local machine and identified as "ds9". Likewise, if the returned handle is omitted from subsequent calls then the respective function will contact the default process.

See Also

`ds9_quit`

6.2 ds9_connect

Synopsis

Establish connection to a DS9 process

Usage

```
Struct_Type = ds9_connect( [xpa_id="ds9", ds9_cl_arg1, ds9_cl_arg2, ... ])
```

Description

Synonym for ds9_launch; more contextually relevant for attaching to a running DS9 process, as opposed to launching an entirely new DS9 process. See ds9_launch for call sequence and more details.

Notes

See Also

`ds9_launch, ds9_quit`

6.3 ds9_quit

Synopsis

Terminate a DS9 process

Usage

```
ds9_quit ( [handle] )
```

Description

Terminates the DS9 process associated with the given handle.

Notes

When handle is omitted the routine will terminate the default process, as described above.

See Also

`ds9_launch`

6.4 ds9_clear

Synopsis

Erase DS9 frame

Usage

```
ds9_clear( [frame_number | "all"][, handle] )
```

Description

Erase the contents of the specified DS9 frame, which by default is the current frame.

See Also

6.5 ds9_center

Synopsis

Center image at position

Usage

```
ds9_center( X, Y, [system="physical" , handle])
```

Description

Shift image so that (X,Y) is positioned at the center of the frame, using the same coordinate system constraints as those described for ds9_get_coords.

Notes

See Also

`ds9_pan, ds9_put_crosshair`

6.6 ds9_get_cmap

Synopsis

Retrieve colormap

Usage

```
(name, inverted) = ds9_get_cmap( [handle] )
```

Description

Retrieve the name of the colormap applied to the current image frame, and an integer value indicating whether it is inverted.

Notes

See Also

`ds9_set_cmap`

6.7 ds9_set_cmap

Synopsis

Change colormap

Usage

```
ds9_set_cmap( "grey|red|..." [, inverted, handle])
```

Description

Redraw the image within the current frame, using the specified colormap. By default the colormap will not be inverted, but that may be controlled by specfying inverted=[0/"no" | 1 / "yes"].

Notes

Unsupported colormap parameters will be ignored.

See Also

`ds9_get_cmap`

6.8 ds9_get_coords

Synopsis

Retrieve position of next mouseclick within any frame

Usage

```
(x,y) = ds9_get_coords( [coord_sys="physical" , handle])
```

Description

Changes cursor to an annulus and pauses DS9 until the next mouseclick within any frame, after which the mouse coordinates are returned. By default these values will be of Double_Type in the physical coordinate system, or (-1, -1) upon error. Image pixel and WCS coordinate values may be obtained by passing in a coord_sys of "image" or "wcs," respectively. Note that the return values may be of String_Type if coord_sys contains additional qualifiers, such as "wcs fk5 sexagesimal".

Notes

Requires DS9 version 3.0b9 or later.

See Also

`ds9_get_crosshair`

6.9 ds9_get_crosshair

Synopsis

Retrieve position of crosshair cursor

Usage

```
(x, y) = ds9_get_crosshair( [coord_sys="physical" , handle])
```

Description

Return the position of the crosshair cursor, under the same coordinate system and return values constraints as those described for ds9_get_coords. (0,0) will be returned when the current frame is not displaying an image. (-1,-1) will be returned upon error (e.g. when no frames exist).

Notes

When an image is loaded into a frame the crosshair cursor will be positioned at its center, even if the crosshair is invisible.

See Also

`ds9_put_crosshair, ds9_get_coords`

6.10 ds9_put_crosshair

Synopsis

Set position of crosshair cursor

Usage

```
ds9_put_crosshair( x, y, [system="physical" , handle])
```

Description

Set the position of the crosshair cursor, using the same coordinate system constraints as those described for ds9_get_coords.

Notes

See Also

`ds9_get_crosshair, ds9_pan`

6.11 ds9_get_version

Synopsis

Retrieve DS9 version information

Usage

```
String = ds9_get_version([handle])
```

Description

Returns a string containing the software version of the DS9 process in use.

Notes

When the DS9 version is obtained directly from a command prompt, say via

linux% xpaget ds9 version

ds9 3.0b9

the application name is included in the version. This function removes that redundancy, so in this instance would return only "3.0b9".

6.12 ds9_get_array

Synopsis

Retrieve displayed image

Usage

```
Array_Type = ds9_get_array( [handle] )
```

Description

Returns the image being displayed in the current DS9 frame, as a 2D S-Lang array of pixel values. The array will be of dimension [Y=NAXIS2, X=NAXIS1] and type reflecting the BITPIX value.

Notes

By default DS9 returns pixel arrays in FITS (big-endian) format. This routine will transparently byteswap its return value, when necessary, to match the the calling platform.

See Also

`ds9_put_array`

6.13 ds9_put_array

Synopsis

Visualize an image pixel array

Usage

```
ds9_put_array(image_pixel_array [, handle] )
```

Description

Transmits an S-Lang array of image pixel values to DS9 for visualization in the current frame. If no frames exist then one will be created first. The array may be either 1D or 2D, and will automatically be tagged with the correct FITS BITPIX and image dimension values.

Notes

A 1D pixel array must contain N^2 elements, and will transmitted to DS9 as an NxN image. DS9 will byteswap all pixel arrays, if necessary, on input.

See Also

ds9_get_array

6.14 ds9_get_file

Synopsis

Retrieve name of file being displayed

Usage

```
String_Type ds9_get_file ( [handle] )
```

Description

Returns the name of file being displayed within the current DS9 frame, which may include a directory path if such was specified to DS9 on input.

Notes

An empty string will be returned if DS9 is not running, has no current frame, or is not displaying a file in the current frame.

See Also

ds9_put_file

6.15 ds9_put_file

Synopsis

Load FITS file

Usage

```
ds9_put_file (FITS_file_name [, handle])
```

Description

Transmits the name of a FITS file to DS9, which it will open and visualize within the current frame. If no frames exist then one will be created first

Notes

The file name may need to include a relative or absolute directory path, if it's located in a directory other than the one from which DS9 was launched.

See Also

ds9_get_file

6.16 ds9_view

Synopsis

Launch DS9 with file or image pixel array

Usage

```
ds9_view( filename | image_pixel_array [, ds9_arg1, ...])
```

Description

A convenience function, which issues a ds9_launch() command followed by ds9_put_file() or ds9_put_array() as appropriate.

Notes

All parameters after the first will be interpreted as DS9 command line arguments.

See Also

`ds9_launch, ds9_put_file, ds9_put_array`

6.17 ds9_get_regions

Synopsis

Retrieve descriptions of regions applied to displayed image

Usage

```
String_Type[] = ds9_get_regions( [file_name | NULL, handle; coord_sys=String ; format=fmt])
```

Description

Returns a possibly empty string array describing the regions that have been applied to the current image frame. Results will be given in the current region file format and coordinate system. The latter defaults to the current region coordinate system in use by DS9, but may be changed changed manually within the Region menu of the GUI or by specifying the named coord_sys qualifier, e.g.

ds9_get_regions("/tmp/my.reg" ; coord_sys="physical")

Note that a semicolon separates named qualifiers from positional parameters.

A string passed in as the first argument will be interpreted as the name of a file to which the region descriptions should be written, and will be echoed back to the caller as the only element within the result array.

Seee the DS9 reference manual for the full range of supported coordinate systems and region file formats.

Notes

Some region formats will prepend one or more comment lines (begining with '#') prior to the actual regions, unless the 'strip' option has been selected within the GUI. Finally, note that the function may be called with no arguments.

See Also

ds9_put_regions

6.18 ds9_put_regions

Synopsis

Request that region descriptions be applied to displayed image

Usage

```
ds9_put_regions( from_file_name | string_array | NULL [, handle ]
```

Description

Transmits a set of region descriptions, from the specified file or string array, to DS9 for application on the current image frame. Pass in NULL to erase all regions.

Notes

File names may need to include a relative or absolute directory path, if the referenced file is located in a directory other than the one from which DS9 was launched.

See Also

ds9_get_regions

6.19 ds9_get_scale

Synopsis

Retrieve image scale

Usage

```
String_Type  ds9_get_scale ( [handle] )
```

Description

Retreive the scale setting of the current image frame, as a string.

Notes

See Also

ds9_set_scale

6.20 ds9_set_scale

Synopsis

Change image scale

Usage

```
ds9_set_scale( "linear|log|sqrt ..." [, handle])
```

Description

Redraw the image within the current frame, using the specified scale.

Notes

Unsupported scale parameters will be ignored.

See Also

`ds9_get_scale`

6.21 ds9_put_wcs

Synopsis

Apply WCS to displayed image, using raw numeric/string values

Usage

```
ds9_put_wcs(crpix, crval, cdelt [, ctype, cunit, alt_wcs_char, handle])
```

Description

Generate a set of FITS world coordinate system header keywords from the given inputs, and transmit them to DS9 for application to the current image frame, replacing any previous WCS keywords of the same name.

Here crpix, crval, cdelt, ctype, and cunit inputs will generally be arrays of length two, with the first three of Float_Type and last two of String_Type. The first element of each array will be used to construct a set of WCS transform keywords for the first image axis, and likewise the set of second elements will apply to the second image axis. If crpix is not an array, or is only of length 1, keywords for the second image axis will be generated from FITS-conformant default values.

Notes

The alt_wcs_char may be one of 'a', ..., 'z', indicating which alternate WCS to use of WCSa, ... WCSz. The module interprets 'p' to mean that the given WCS should be viewed by DS9 as the physical coordinate system.

See Also

`ds9_put_wcs_keys, ds9_put_wcs_struct, ds9_wcs_edit`

6.22 ds9_put_wcs_keys

Synopsis

Apply WCS to displayed image, using pre-formatted FITS keywords

Usage

```
ds9_put_wcs_keys( from_file_name | string_array [, handle])
```

Description

Transmit a set of FITS world coordinate system header keywords to DS9, for application to the current image frame, replacing any previous WCS keywords of the same name.

Notes

Keyword name/value pairs may be formatted either in accordance with the FITS 80 character card standard, or simply separated by whitespace.

See Also

`ds9_put_wcs , ds9_put_wcs_struct, ds9_wcs_edit`

6.23 ds9_put_wcs_struct

Synopsis

Apply WCS to displayed image, using structure field values

Usage

```
ds9_put_wcs_struct(struct [, alt_wcs_char, handle])
```

Description

Like ds9_put_wcs(), except here the raw crpix, crval, cdelt, ctype, and cunit values will be taken from fields of the same name within the specified structure.

See Also

`ds9_put_wcs , ds9_put_wcs_keys, ds9_wcs_edit`

6.24 ds9_pan

Synopsis

Shift image position

Usage

```
ds9_pan( X, Y, [system="physical" , handle])
```

Description

Pan image by X pixels horizontally and Y pixels vertically, using the same coordinate system constraints as those described for ds9_get_coords.

Notes

X and Y may be negative.

See Also

`ds9_center, ds9_put_crosshair`

6.25 ds9_get_zoom

Synopsis

Retreive zoom level

Usage

```
Double_Type  ds9_get_zoom ( [handle] )
```

Description

Retrieve zoom level of current image frame, as a Double_Type value.

Notes

See Also

`ds9_center, ds9_pan, ds9_set_zoom`

6.26 ds9_set_zoom

Synopsis

Zoom in/out

Usage

```
ds9_set_zoom( zoom_factor [, handle])
```

Description

Set current zoom to specified value. Values greater than 1 zoom in to image features, while values less than 1 zoom out.

Notes

See Also

`ds9_center, ds9_pan, ds9_get_zoom`

6.27 ds9_send

Synopsis

Send arbitrary XPA command strings to DS9

Usage

```
ds9_send( ds9_xpa_command_string [, handle])
```

Description

This function provides a useful catch-all control mechanism, by allowing arbitrary XPA command strings to be sent to DS9. For example, at the time of this writing the DS9 module did not provide a high-level interface, through which various scaling factors can be set, such as ds9_scale("log"). Instead, the DS9 scale factor can be customized with the command ds9_send("scale log").

Notes

Use this function when the given XPA command is not expected to return a result to the caller. Use ds9_recv() when issuing XPA commands which are expected to return a result,

See Also

`ds9_recv`

6.28 ds9_recv

Synopsis

Send arbitrary XPA command strings to DS9 and return a result

Usage

```
result = ds9_recv( ds9_xpa_command_string [, handle])
```

Description

This is another catch-all function like ds9_send(), only it allows a result to be returned from DS9 to the caller.

Notes

Many of the higher level DS9 module functions reformat the information returned by DS9 into a more useful form before returning it to the caller. Because it is more generic, ds9_recv() DOES NOT do this. For example, suppose the zoom level in the DS9 GUI is 1. In this case the ds9_get_zoom() function would return a floating point value of 1, but ds9_recv("zoom") would return the string "1\n".

See Also

`ds9_send`

6.29 ds9_wcs_edit

Synopsis

Interactive WCS editor

Usage

```
ds9_wcs_edit()
```

Description

Launches a guilet from which WCS values may be applied to the current image.

Notes

The loading of this function is deferred until it is actually called, because it requires that the SLgtk module be installed on your system. When the function is first invoked the intepreter will also attempt to load SLgtk (if it has not already been loaded); if that fails this function will not be executed.

See Also

`ds9_put_wcs, ds9_put_wcs_keys, ds9_put_wcs_struct`

6.30 ds9_new

Synopsis

Establish connection to a DS9 process, returning an OO-like object

Usage

```
Struct_Type = ds9_new( [xpa_id="ds9", ds9_cl_arg1, ds9_cl_arg2, ... ])
```

Description

This function is similar to ds9_launch(), only instead of returning a handle it returns a structure which may subsequently be used in an object-oriented fashion. For example, consider

variable d = ds9_new(); d.put_array([1:100*100]);

Notes

This function accepts the same arguments as does ds9_launch().

See Also

`ds9_launch`