Numdisplay Documentation

Release 1.6.1 (25-Mar-2011)

Warren Hack

Jan 30, 2018

Contents

Contents:

CHAPTER 1

NumDisplay Class Interface

This class provides the primary functionality for displaying numpy arrays in DS9 using the IIS protocol.

numdisplay: Package for displaying numpy arrays in IRAF-compatible image display tool such as DS9 or XIMTOOL.

This package provides several methods for controlling the display of the numpy array; namely,

open(imtdev=None):: Open the default display device or the device specified in 'imtdev', such as 'inet:5137' or 'fifo:/dev/imt1o'.

close():: Close the display device defined by 'imtdev'. This must be done before resetting the display buffer to a new size.

display(pix, name=None, bufname=None, z1=None, z2=None, quiet=False, transform=None, scale=None, offset=None, fra
Display the scaled array in display tool (ds9/ximtool/...).

readcursor(sample=0):: Return a single cursor position from the image display. By default, this operation will wait for a keystroke before returning the cursor position. If 'sample' is set to 1, then it will NOT wait to read the cursor. This will return a string containing: x,y,frame and key.

help():: print Version ID and this message.

Notes

Displaying a numpy array object involves:

1. Opening the connection to a usable display tool (such as DS9).
2. Setting the display parameters for the array, such as min and max array value to be used for min and max grey scale level, along with any offset, scale factor and/or transformation function to be applied to the array.
3. Applying any transformation to the input array. This transformation could be a simple numpy ufunc or a user-defined function that returns a modified array.
4. Building the byte-scaled version of the transformed array and sending it to the display tool. The image sent to the display device will be trimmed to fit the image buffer defined by the 'imtdev' device from the

'imtoolrc' or the 'stdimage' variable under IRAF. If the image is smaller than the buffer, it will be centered in the display device.

All pixel positions reported back will be relative to the full image size.

Examples

The user starts with a 1024x1024 array in the variable 'fdata'. This array has min pixel value of -157.04 and a max pixel value of 111292.02. The display tool DS9 has already been started from the host level and awaits the array for display. Displaying the array requires:

```
>>> import numdisplay
>>> numdisplay.display(fdata)
```

If there is a problem connecting to the DS9 application, the connection can be manually started using:

```
>>> numdisplay.open()
```

To bring out the fainter features, an offset value of 158 can be added to the array to allow a 'log' scaling can be applied to the array values using:

```
>>> numdisplay.display(fdata,transform=numpy.log,offset=158.0)
```

To redisplay the image with default full-range scaling:

```
>>> numdisplay.display(fdata)
```

To redisplay using the IRAF display zscale algorithm, and with a contrast value steeper than the default value of 0.25:

```
>>> numdisplay.display(fdata, zscale=True, contrast=0.5)
```

class stsci.numdisplay.**NumDisplay**

Class to manage the attributes and methods necessary for displaying the array in the image display tool.

This class contains the methods: open(imtdev=None):

close():

set(z1=None,z2=None,scale=None,factor=None,frame=None): reset(names=None)

display(pix, name=None, bufname=None):

readcursor():

checkDisplay()

close()

Close the display device entry.

display(*pix*, *name=None*, *bufname=None*, *z1=None*, *z2=None*, *transform=None*, *zscale=False*, *contrast=0.25*, *scale=None*, *offset=None*, *frame=None*, *quiet=False*)

Displays byte-scaled (UInt8) n to XIMTOOL device. This method uses the IIS protocol for displaying the data to the image display device, which requires the data to be byte-scaled.

If input is not byte-scaled, it will perform scaling using set values/defaults.

Parameters

- **name** (*str*) – optional name to pass along for identifying array

- **bufname** (*str*) – name of buffer to use for displaying array (such as 'imt512'). Other valid values include:

```
'iraf': look for 'stdimage' and use that buffer or default to
↪'imt1024' [1024x1024 buffer]
None  : ignore 'stdimage' and automatically select a buffer␣
↪matched to the size of the image.
```

- **z1,z2** (*float*) – minimum/maximum pixel value to display. Not specifying values will default to the full range values of the input array.
- **transform** (*function*) – Python function to apply to array (function)
- **zscale** (*bool*) – Specify whether or not to use an algorithm like that in the IRAF display task. If zscale=True, any z1 and z2 set in the call to display are ignored. Using zscale=True invalidates any transform specified in the call.
- **contrast** (*float (Default=0.25)*) – Same as the *contrast* parameter in the IRAF *display* task. Only applies if zscale=True. Higher contrast values make z1 and z2 closer together, while lower values give a gentler (wider) range.
- **scale** (*float/int*) – multiplicative scale factor to apply to array. The value of this parameter remains persistent, so to reset it you must specify scale=1 in the display call.
- **offset** (*float/int*) – additive factor to apply to array before scaling. This value is persistent, so to reset it you have to set it to 0.
- **frame** (*int*) – image buffer frame number in which to display array
- **quiet** (*bool (Default: False)*) – if True, this parameter will turn off all status messages

Notes

The display parameters set here will ONLY apply to the display of the current array.

getHandle()

open(*imtdev=None*)
Open a display device.

readcursor(*sample=0*)
Return the cursor position from the image display.

reset(*names=None*)
Reset specific attributes, or all by default.

Parameters **names** (*string or list of strings*) – names of attributes to be reset, separated by commas or spaces; the default is to reset all attributes to None

set(*frame=None*, *z1=None*, *z2=None*, *contrast=None*, *transform=None*, *scale=None*, *offset=None*)
Allows user to set multiple parameters at one time.

CHAPTER 2

DisplayDev module

displaydev.py: Interact with IRAF-compatible image display

Modeled after the NOAO Client Display Library (CDL)

Public functions:

readCursor(sample=0) Read image cursor position

open(imtdev=None) Open a connection to the display server. This is called automatically by readCursor if the display has not already been opened, so it is not generally necessary for users to call it.

See the open doc string for info on the imtdev argument, which allows various forms of socket and network connections.

close() Close the active display server. Called automatically on exit.

Various classes are defined for the different connections (ImageDisplay, ImageDisplayProxy, UnixImageDisplay, InetImageDisplay, FifoImageDisplay). They should generally be created using the _open factory function. This could be used to maintain references to multiple display servers.

Ultimately more functionality may be added to make this a complete replacement for CDL.

Id

class stsci.numdisplay.displaydev.**FifoImageDisplay**(*infile*, *outfile*)
FIFO version of image display

_write(*s*)
Write string s to image display

Raises IOError on failure

class stsci.numdisplay.displaydev.**ImageDisplay**
Interface to IRAF-compatible image display

_read(*n*)
Read n bytes from image display and return as string

Raises IOError on failure. If a Tkinter widget exists, runs a Tk mainloop while waiting for data so that the Tk widgets remain responsive.

_write(*s*)
Write string s to image display

Raises IOError on failure

_writeHeader(*tid*, *subunit*, *thingct*, *x*, *y*, *z*, *t*)
Write request to image display

close(*os_close=<built-in function close>*)
Close image display connection

eraseFrame()
Sends commands to erase active frame.

getConfigno(*stdname*)
Determine which config number matches specified frame buffer name.

readCursor(*sample=0*)
Read image cursor value for this image display

Return immediately if sample is true, or wait for keystroke if sample is false (default). Returns a string with x, y, frame, and key.

readData(*x*, *y*, *pix*)
Reads data from x,y position in active frame.

readInfo()
Read tx and ty from active frame of display device.

readWCS(*wcsinfo*)
Reads WCS information from active frame of display device.

selectFB(*nx*, *ny*, *reset=None*, *useiraf=True*)
Select the frame buffer that best matches the input image size.

setCursor(*x*, *y*, *wcs*)
Moves cursor to specified position in frame.

setFBconfig(*fbnum*, *bufname=None*)
Set the frame buffer values for the given frame buffer name.

setFrame(*frame_num=1*)
Sets the active frame in frame buffer to specified value.

syncWCS(*wcsinfo*)
Update WCS to match frame buffer being used.

writeData(*x*, *y*, *pix*)
Writes out image data to x,y position in active frame.

writeImage(*pix*, *wcsinfo*)
Write out image to display device in 32Kb sections.

writeWCS(*wcsinfo*)
Writes out WCS information for frame to display device.

class stsci.numdisplay.displaydev.**ImageDisplayProxy**(*imtdev=None*)
Interface to IRAF-compatible image display

This is a proxy to the actual display that allows retries on failures and can switch between display connections.

checkDisplay()
Returns True if a valid connection to a display device is found, False if no connection could be found.

close()
Close active image display connection

open(*imtdev=None*)
Open image display connection, closing any active connection

readCursor(*sample=0*)
Read image cursor value for the active image display

Return immediately if sample is true, or wait for keystroke if sample is false (default). Returns a string with x, y, frame, and key. Opens image display if necessary.

class stsci.numdisplay.displaydev.**InetImageDisplay**(*port*, *hostname=None*)
INET socket version of image display

class stsci.numdisplay.displaydev.**UnixImageDisplay**(*filename*, *family=<AddressFamily.AF_UNIX: 1>*, *type=<SocketKind.SOCK_STREAM: 1>*)
Unix socket version of image display

close()
Close image display connection

stsci.numdisplay.displaydev.**_open**(*imtdev=None*)
Open connection to the image display server

This is a factory function that returns an instance of the ImageDisplay class for the specified imtdev. The default connection if no imtdev is specified is given in the environment variable IMTDEV (if defined) or is "unix:/tmp/.IMT%d". Failing that, a connection is attempted on the /dev/imt1[io] named fifo pipes.

The syntax for the imtdev argument is <domain>:<address> where <domain> is one of "inet" (internet tcp/ip socket), "unix" (unix domain socket) or "fifo" (named pipe). The form of the address depends upon the domain, as illustrated in the examples below.

```
inet:5137                   Server connection to port 5137 on the local
                            host.  For a client, a connection to the
                            given port on the local host.

inet:5137:foo.bar.edu       Client connection to port 5137 on internet
                            host foo.bar.edu.  The dotted form of address
                            may also be used.

unix:/tmp/.IMT212           Unix domain socket with the given pathname
                            IPC method, local host only.

fifo:/dev/imt1i:/dev/imt1o FIFO or named pipe with the given pathname.
                            IPC method, local host only.  Two pathnames
                            are required, one for input and one for
                            output, since FIFOs are not bidirectional.
                            For a client the first fifo listed will be
                            the client's input fifo; for a server the
                            first fifo will be the server's output fifo.
                            This allows the same address to be used for
                            both the client and the server, as for the
                            other domains.
```

The address field may contain one or more "%d" fields. If present, the user's UID will be substituted (e.g. "unix:/tmp/.IMT%d").

CHAPTER 3

zscale module

stsci.numdisplay.zscale.**zscale**(*image*, *nsamples=1000*, *contrast=0.25*, *bpmask=None*, *zmask=None*)

Implement IRAF zscale algorithm

Parameters

- **image** (*arr*) – 2-d numpy array
- **nsamples** (*int (Default: 1000)*) – Number of points in array to sample for determining scaling factors
- **contrast** (*float (Default: 0.25)*) – Scaling factor for determining min and max. Larger values increase the difference between min and max values used for display.
- **bpmask** (*None*) – Not used at this time
- **zmask** (*None*) – Not used at this time

Returns

Return type (z1, z2)

CHAPTER 4

overlay module

stsci.numdisplay.overlay.**circle**(***kwargs*)
Draw a circle.

Parameters

- **x** (*int*) – image X coordinate of center
- **y** (*int*) – image Y coordinate of center
- **center** (*tuple*) – (x,y) coordinates of center
- **radius** (*int*) – radius of circle
- **color** (*int*) – color code to use; if not specified, use default
- **undo** (*bool*) – if specified [default=True], keep track of overlays for undo()

Examples

Samples illustrating the syntax include:

```
overlay.circle (x=x0, y=y0, radius=r)
overlay.circle (center=(x0,y0), radius=r)
overlay.circle (x=x0, y=y0, radius=r, color=overlay.C_<color>)
```

stsci.numdisplay.overlay.**close_display**(*frame=1*)
Close the device.

stsci.numdisplay.overlay.**marker**(***kwargs*)
Draw a character.

Parameters

- **x** (*int*) – image X coordinate of point
- **y** (*int*) – image Y coordinate of point
- **mark** (*str*) – character to be drawn

- **size** (*int*) – magnification to be used in drawing the character
- **color** (*int*) – color code to use; if not specified, use default
- **undo** (*bool*) – if specified [default=True], keep track of overlays for undo()

Examples

Samples illustrating the syntax include:

```
overlay.marker (x=x0, y=y0, mark='+')
overlay.marker (x=x0, y=y0, mark='+', size=2)
overlay.marker (x=x0, y=y0, mark='+', color=overlay.C_<color>)
```

stsci.numdisplay.overlay.**point**(***kwargs*)

Draw a point.

Parameters

- **x** (*int*) – image X coordinate of point
- **y** (*int*) – image Y coordinate of point
- **center** (*tuple*) – (x,y) coordinates of point
- **color** (*int*) – color code to use; if not specified, use default
- **undo** (*bool*) – if specified [default=True], keep track of overlays for undo()

Examples

Samples illustrating the syntax include:

```
overlay.point (x=x0, y=y0)
overlay.point (center=(x0,y0))
overlay.point (x=x0, y=y0, color=overlay.C_<color>)
```

stsci.numdisplay.overlay.**polyline**(***kwargs*)

Draw a series of connected line segments.

Parameters

- **points** (*list of tuples*) – (x,y) points to connect with line segments
- **vertices** (*list of tuples*) – (x,y) points to connect with line segments
- **color** (*int*) – color code to use; if not specified, use default
- **undo** (*bool*) – if specified [default=True], keep track of overlays for undo()

Examples

Samples illustrating the syntax include:

```
overlay.polyline (points=[(x1,y1), (x2,y2), (x3,y3)])
overlay.polyline (vertices=[(x1,y1), (x2,y2), (x3,y3)])
overlay.polyline (points=[(x1,y1), (x2,y2), (x3,y3)],
                  color=overlay.C_<color>)
```

stsci.numdisplay.overlay.**rectangle**(***kwargs*)
Draw a rectangle.

Parameters

- **left** (*int*) – image X coordinate of left edge
- **right** (*int*) – image X coordinate of right edge
- **lower** (*int*) – image Y coordinate of lower edge
- **upper** (*int*) – image Y coordinate of upper edge
- **center** (*tuple*) – (x,y) coordinates of middle of rectangle
- **width** (*int*) – width of rectangle (X direction)
- **height** (*int*) – height of rectangle (Y direction)
- **color** (*int*) – color code to use; if not specified, use default
- **undo** (*bool*) – if specified [default=True], keep track of overlays for undo()

Examples

Samples illustrating the syntax include:

```
overlay.rectangle (left=x1, right=x2, lower=y1, upper=y2)
overlay.rectangle (center=(x0,y0), width=w, height=h)
overlay.rectangle (left=x1, lower=y1, center=(x0,y0))
overlay.rectangle (left=x1, lower=y1, width=w, height=h)
overlay.rectangle (right=x2, upper=y2, width=w, height=h)
overlay.rectangle (right=x2, upper=y2, center=(x0,y0))
overlay.rectangle (left=x1, right=x2, lower=y1, upper=y2, color=overlay.C_<color>)
```

stsci.numdisplay.overlay.**set**(*color=None*, *radius=None*)
Specify the color or the radius.

Parameters

- **color** (*int*) – color code to use for graphic overlays; the allowed values (202..217) are:

```
C_BLACK, C_WHITE, C_RED, C_GREEN, C_BLUE, C_YELLOW, C_CYAN, C_
↪MAGENTA,
C_CORAL, C_MAROON, C_ORANGE, C_KHAKI, C_ORCHID, C_TURQUOISE, C_
↪VIOLET,
C_WHEAT
```

- **radius** (*int*) – radius to use when drawing circles

stsci.numdisplay.overlay.**undo**()
Restore the values before the last overlay was written.

CHAPTER 5

ichar module

stsci.numdisplay.ichar.**expandchar**(*indices*, *size*)
block replicate a character. This implementation is inefficient but probably won't matter unless many large characters are used

stsci.numdisplay.ichar.**initichar**()
read the data file (old form) and generate the dict for numdisplay to use

stsci.numdisplay.ichar.**initichar_old**()
read the data file (old form) and generate the dict for numdisplay to use

stsci.numdisplay.ichar.**nextchar**(*lines*)
read next character data in file, and return numeric array 5x7 pixels

stsci.numdisplay.ichar.**read_inc**()
Read the iraf include file that defines the bit patterns for characters used for image display

CHAPTER 6

imconfig module

Version 1.0alpha - 9-Oct-2003 (WJH)

loadImtoolrc (imtoolrc=None): Locates, then reads in IMTOOLRC configuration file from system or user-specified location, and returns the dictionary for reference.

The table gets loaded into a dictionary of the form: {configno:{'nframes':n,'width':nx,'height':ny},...}

It can then be accessed using the syntax: fbtab[configno][attribute]

For example: fbtab = loadImtoolrc() print fbtab[34]['width'] 1056 1024

stsci.numdisplay.imconfig.**loadImtoolrc**(*imtoolrc=None*)
Locates, then reads in IMTOOLRC configuration file from system or user-specified location, and returns the dictionary for reference.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

C

D

E

F

G

I

L

M

N

O

P

R

S

U

W

Z