API Documentation

March 14, 2008

Contents

1 Package numdisplay

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

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.

    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,
                transform=None, scale=None, offset=None, frame=None):
            Display the scaled array in display tool (ds9/ximtool/...).
            name -- optional name to pass along for identifying array

            bufname -- name of buffer to use for displaying array
                        to best match size of array (such as 'imt1024')
                        [default: 512x512 buffer named 'imt512']

            z1,z2  -- minimum/maximum pixel value to display (float)
                      Not specifying values will default
                      to the full range values of the input array.

            transform -- Python function to apply to array (function)
            zscale -- 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 -- same as the contrast parameter in the IRAF display
                        task.  Only applies if zscale=True.  Default value = 0.25.
                        Higher contrast values make z1 and z2 closer together,
                        while lower values give a gentler (wider) range.

            scale  -- multiplicative scale factor to apply to array (float/int)
                      Persistent, so to reset it you must specify scale=1 in the
                      display call.

            offset -- additive factor to apply to array before scaling (float/int)
                      This value is persistent, so to reset it you have to set it
                      to 0.

            frame  -- image buffer frame number in which to display array
                        (integer)

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

        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.

Example:
    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)
```

1.1 Modules

1.2 Functions

help()

Print out doc string with syntax and example.

1.3 Class NumDisplay

__builtin__.object ─┐

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():
```

1.3.1 Methods

__init__(*self*)
Overrides: __builtin__.object.__init__

---

close(*self*)

Close the display device entry.

---

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

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.

---

open(*self*, *imtdev*=`None`)

Open a display device.

---

readcursor(*self*, *sample*=`0`)

Return the cursor position from the image display.

---

reset(*self*, *names*=`None`)

Reset specific attributes, or all by default.

Parameters

`names`: names of attributes to be reset, separated by commas or spaces; the default is to reset all attributes to None
(type=string or list of strings)

---

set(*self*, *frame*=`None`, *z1*=`None`, *z2*=`None`, *contrast*=`None`, *transform*=`None`, *scale*=`None`, *offset*=`None`)

Allows user to set multiple parameters at one time.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

2 Module numdisplay.displaydev

```
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: displaydev.py 1064 2007-12-28 18:50:11Z hodge $
```

2.1 Functions

help()

2.2 Class FifoImageDisplay

__builtin__.object

numdisplay.displaydev.ImageDisplay

FifoImageDisplay

FIFO version of image display

2.2.1 Methods

__init__(*self*, *infile*, *outfile*)
Overrides: numdisplay.displaydev.ImageDisplay.__init__

__del__(*self*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from ImageDisplay: close, eraseFrame, getConfigno, readCursor, readData, readInfo, readWCS, selectFB, setCursor, setFBconfig, setFrame, syncWCS, writeData, writeImage, writeWCS

2.3 Class ImageDisplay

```
__builtin__.object ─┐
                    ImageDisplay
```

Known Subclasses: FifoImageDisplay, ImageDisplayProxy, UnixImageDisplay

Interface to IRAF-compatible image display

2.3.1 Methods

__init__(*self*)
Overrides: __builtin__.object.__init__

close(*self*, *os_close*=`<built-in function close>`)

Close image display connection

eraseFrame(*self*)

Sends commands to erase active frame.

getConfigno(*self*, *stdname*)

Determine which config number matches specified frame buffer name.

readCursor(*self*, *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(*self*, *x*, *y*, *pix*)

Reads data from x,y position in active frame.

readInfo(*self*)

Read tx and ty from active frame of display device.

readWCS(*self*, *wcsinfo*)

Reads WCS information from active frame of display device.

selectFB(*self*, *nx*, *ny*, *reset*=`None`)

Select the frame buffer that best matches the input image size.

setCursor(*self*, *x*, *y*, *wcs*)

Moves cursor to specified position in frame.

setFBconfig(*self*, *fbnum*, *bufname*=`None`)

Set the frame buffer values for the given frame buffer name.

setFrame(*self*, *frame_num*=`1`)

Sets the active frame in frame buffer to specified value.

syncWCS(*self*, *wcsinfo*)

Update WCS to match frame buffer being used.

writeData(*self*, *x*, *y*, *pix*)

Writes out image data to x,y position in active frame.

writeImage(*self*, *pix*, *wcsinfo*)

Write out image to display device in 32Kb sections.

writeWCS(*self*, *wcsinfo*)

Writes out WCS information for frame to display device.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

2.4 Class ImageDisplayProxy

__builtin__.object

numdisplay.displaydev.ImageDisplay

ImageDisplayProxy

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.

2.4.1 Methods

__init__(*self*, *imtdev*=`None`)
Overrides: numdisplay.displaydev.ImageDisplay.__init__

close(*self*)

Close active image display connection

Overrides: numdisplay.displaydev.ImageDisplay.close

open(*self*, *imtdev*=`None`)

Open image display connection, closing any active connection

readCursor(*self*, *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.

Overrides: numdisplay.displaydev.ImageDisplay.readCursor

setCursor(*self*, *x*, *y*, *wcs*)

Moves cursor to specified position in frame.

Overrides: numdisplay.displaydev.ImageDisplay.setCursor extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from ImageDisplay: eraseFrame, getConfigno, readData, readInfo, readWCS, selectFB, setFBconfig, setFrame, syncWCS, writeData, writeImage, writeWCS

2.5 Class ImageWCS

__builtin__.object
ImageWCS

2.5.1 Methods

__init__(*self*, *pix*, *name*=`None`, *title*=`None`, *z1*=`None`, *z2*=`None`)
Overrides: __builtin__.object.__init__

__str__(*self*)
Overrides: __builtin__.object.__str__

update(*self*, *wcsstr*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__

2.6 Class InetImageDisplay

```
__builtin__.object ─┐
                    │
numdisplay.displaydev.ImageDisplay ─┐
                                    │
 numdisplay.displaydev.UnixImageDisplay ─┐
                                         │
                                  InetImageDisplay
```

INET socket version of image display

2.6.1 Methods

__init__(*self*, *port*, *hostname*=`None`)
Overrides: numdisplay.displaydev.UnixImageDisplay.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from ImageDisplay: eraseFrame, getConfigno, readCursor, readData, readInfo, readWCS, selectFB, setCursor, setFBconfig, setFrame, syncWCS, writeData, writeImage, writeWCS
Inherited from UnixImageDisplay: close

2.7 Class UnixImageDisplay

```
__builtin__.object ─┐
                    │
numdisplay.displaydev.ImageDisplay ─┐
                                    │
                             UnixImageDisplay
```

Known Subclasses: InetImageDisplay

Unix socket version of image display

2.7.1 Methods

__init__(*self*, *filename*, *family*=`1`, *type*=`1`)
Overrides: numdisplay.displaydev.ImageDisplay.__init__

close(*self*)

Close image display connection

Overrides: numdisplay.displaydev.ImageDisplay.close

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

Inherited from ImageDisplay: eraseFrame, getConfigno, readCursor, readData, readInfo, readWCS, selectFB, setCursor, setFBconfig, setFrame, syncWCS, writeData, writeImage, writeWCS

3 Module numdisplay.imconfig

```
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
```

3.1 Functions

help()

loadImtoolrc(*imtoolrc*=`None`)

Locates, then reads in IMTOOLRC configuration file from system or user-specified location, and returns the dictionary for reference.

4 Module numdisplay.setup

4.1 Class smart_install_data

distutils.cmd.Command ─┐

distutils.command.install_data.install_data ─┐

smart_install_data

4.1.1 Methods

run(*self*)

A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.
This method must be implemented by all command classes.

Overrides: distutils.command.install_data.install_data.run extit(inherited documentation)

Inherited from Command: __init__, __getattr__, announce, copy_file, copy_tree, debug_print, dump_options, ensure_dirname, ensure_filename, ensure_finalized, ensure_string, ensure_string_list, execute, get_command_name, get_finalized_command, get_sub_commands, make_archive, make_file, mkpath, move_file, reinitialize_command, run_command, set_undefined_options, spawn, warn
Inherited from install_data: finalize_options, get_inputs, get_outputs, initialize_options

4.1.2 Class Variables

Name	Description
Inherited from Command: sub_commands *(p. ??)*	
Inherited from install_data: boolean_options *(p. ??)*, description *(p. ??)*, user_options *(p. ??)*	

5 Module numdisplay.zscale

5.1 Functions

zsc_compute_sigma(*flat*, *badpix*, *npix*)

zsc_fit_line(*samples*, *npix*, *krej*, *ngrow*, *maxiter*)

zsc_sample(*image*, *maxpix*, *bpmask*=`None`, *zmask*=`None`)

zscale(*image*, *nsamples*=`1000`, *contrast*=`0.25`, *bpmask*=`None`, *zmask*=`None`)

Implement IRAF zscale algorithm nsamples=1000 and contrast=0.25 are the IRAF display task defaults bpmask and zmask not implemented yet image is a 2-d numpy array returns (z1, z2)

Index