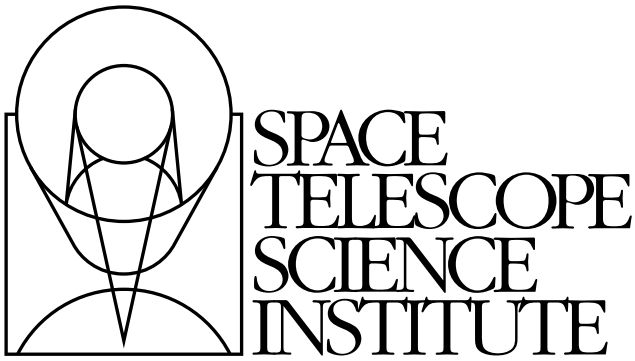

pywcs Documentation

Release 1.12.1.dev5778

Michael Droettboom

June 23, 2016

Python wrappers, SIP, Paper IV support (BSD License):

Michael Droettboom, Nadia Dencheva

Space Telescope Science Institute

wcslib (LGPL License):

Mark Calabretta

Australia Telescope National Facility, CSIRO (wcslib)

Pywcs provides transformations following the SIP conventions, Paper IV table lookup distortion, and the core WCS functionality provided by wcslib. Each of these transformations can be used independently or together in a standard pipeline.

The basic workflow is as follows:

1. `import pywcs`
2. Call the *`pywcs.WCS`* constructor with a pyfits header and/or hdulist object.
3. Optionally, if the FITS file uses any deprecated or non-standard features, you may need to call one of the `fix` methods on the object.
4. Use one of the following transformation methods:
   - *`all_pix2sky`*: Perform all three transformations from pixel to sky coordinates.
   - *`wcs_pix2sky`*: Perform just the core WCS transformation from pixel to sky coordinates.
   - *`wcs_sky2pix`*: Perform just the core WCS transformation from sky to pixel coordinates.
   - *`sip_pix2foc`*: Convert from pixel to focal plane coordinates using the SIP polynomial coefficients.
   - *`sip_foc2pix`*: Convert from focal plane to pixel coordinates using the SIP polynomial coefficients.
   - *`p4_pix2foc`*: Convert from pixel to focal plane coordinates using the table lookup distortion method described in Paper IV.
   - *`det2im`*: Convert from detector coordinates to image coordinates. Commonly used for narrow column correction.

Contents:

CHAPTER

ONE

EXAMPLE USAGE

1.1 Loading WCS information from a FITS file

This example loads a FITS file (supplied on the commandline) and uses the WCS cards in its primary header to transform.

```
# Load the WCS information from a fits header, and use it
# to convert pixel coordinates to world coordinates.

from __future__ import division, print_function # confidence high

import numpy
import pywcs
from astropy.io import fits as pyfits
import sys

# Load the FITS hdulist using pyfits
hdulist = pyfits.open(sys.argv[-1])

# Parse the WCS keywords in the primary HDU
wcs = pywcs.WCS(hdulist[0].header)

# Print out the "name" of the WCS, as defined in the FITS header
print(wcs.wcs.name)

# Print out all of the settings that were parsed from the header
wcs.wcs.print_contents()

# Some pixel coordinates of interest.
pixcrd = numpy.array([[0,0],[24,38],[45,98]], numpy.float_)

# Convert pixel coordinates to world coordinates
# The second argument is "origin" -- in this case we're declaring we
# have 1-based (Fortran-like) coordinates.
sky = wcs.wcs_pix2sky(pixcrd, 1)
print(sky)

# Convert the same coordinates back to pixel coordinates.
pixcrd2 = wcs.wcs_sky2pix(sky, 1)
print(pixcrd2)

# These should be the same as the original pixel coordinates, modulo
# some floating-point error.
assert numpy.max(numpy.abs(pixcrd - pixcrd2)) < 1e-6
```

1.2 Building a WCS structure programmatically

This example, rather than starting from a FITS header, sets WCS values programmatically, uses those settings to transform some points, and then saves those settings to a new FITS header.

```
# Set the WCS information manually by setting properties of the WCS
# object.

from __future__ import division, print_function # confidence high

import numpy
import pywcs
from astropy.io import fits as pyfits
import sys

# Create a new WCS object.  The number of axes must be set
# from the start
wcs = pywcs.WCS(naxis=2)

# Set up an "Airy's zenithal" projection
# Vector properties may be set with Python lists, or Numpy arrays
wcs.wcs.crpix = [-234.75, 8.3393]
wcs.wcs.cdelt = numpy.array([-0.066667, 0.066667])
wcs.wcs.crval = [0, -90]
wcs.wcs.ctype = ["RA---AIR", "DEC--AIR"]
wcs.wcs.set_pv([(2, 1, 45.0)])

# Print out the "name" of the WCS, as defined in the FITS header
print(wcs.wcs.name)

wcs.wcs.print_contents()

# Some pixel coordinates of interest.
pixcrd = numpy.array([[0,0],[24,38],[45,98]], numpy.float_)

# Convert pixel coordinates to world coordinates
world = wcs.wcs_pix2sky(pixcrd, 1)
print(world)

# Convert the same coordinates back to pixel coordinates.
pixcrd2 = wcs.wcs_sky2pix(world, 1)
print(pixcrd2)

# These should be the same as the original pixel coordinates, modulo
# some floating-point error.
assert numpy.max(numpy.abs(pixcrd - pixcrd2)) < 1e-6

# Now, write out the WCS object as a FITS header
header = wcs.to_header()

# header is a PyFITS Header object.  We can use it to create a new
# PrimaryHDU and write it to a file.
hdu = pyfits.PrimaryHDU(header=header)
hdu.writeto('test.fits')
```

CHAPTER

TWO

API DOCUMENTATION

2.1 pywcs

Pywcs provides transformations following the SIP conventions, Paper IV table lookup distortion, and the core WCS functionality provided by wcslib. Each of these transformations can be used independently or together in a standard pipeline.

The basic workflow is as follows:

1. `import pywcs`
2. Call the *pywcs.WCS* constructor with a pyfits header and/or hdulist object.
3. Optionally, if the FITS file uses any deprecated or non-standard features, you may need to call one of the `fix` methods on the object.
4. Use one of the following transformation methods:
   - *all_pix2sky*: Perform all three transformations from pixel to sky coordinates.
   - *wcs_pix2sky*: Perform just the core WCS transformation from pixel to sky coordinates.
   - *wcs_sky2pix*: Perform just the core WCS transformation from sky to pixel coordinates.
   - *sip_pix2foc*: Convert from pixel to focal plane coordinates using the SIP polynomial coefficients.
   - *sip_foc2pix*: Convert from focal plane to pixel coordinates using the SIP polynomial coefficients.
   - *p4_pix2foc*: Convert from pixel to focal plane coordinates using the table lookup distortion method described in Paper IV.
   - *det2im*: Convert from detector coordinates to image coordinates. Commonly used for narrow column correction.

2.2 Classes

2.2.1 WCS

class `pywcs.`**WCS**(*header=None*, *fobj=None*, *key=' '*, *minerr=0.0*, *relax=False*, *naxis=None*, *keysel=None*, *colsel=None*, *fix=True*)

WCS objects perform standard WCS transformations, and correct for SIP and Paper IV table-lookup distortions, based on the WCS keywords and supplementary data read from a FITS file.

- *header*: A string containing the header content, or a PyFITS header object. If *header* is not provided or None, the object will be initialized to default values.

- *fobj*: A PyFITS file (hdulist) object. It is needed when header keywords point to a Paper IV Lookup table distortion stored in a different extension.
- *key*: A string. The name of a particular WCS transform to use. This may be either `' '` or `'A'-'Z'` and corresponds to the `"a"` part of the `CTYPEia` cards. *key* may only be provided if *header* is also provided.
- *minerr*: A floating-point value. The minimum value a distortion correction must have in order to be applied. If the value of `CQERRja` is smaller than *minerr*, the corresponding distortion is not applied.
- *relax*: Degree of permissiveness:
  - `False`: Recognize only FITS keywords defined by the published WCS standard.
  - `True`: Admit all recognized informal extensions of the WCS standard.
  - `int`: a bit field selecting specific extensions to accept. See *Header-reading relaxation constants* for details.
- *naxis*: int or sequence. Extracts specific coordinate axes using `sub()`. If a header is provided, and *naxis* is not `None`, *naxis* will be passed to `sub()` in order to select specific axes from the header. See `sub()` for more details about this parameter.
- *keysel*: A list of flags used to select the keyword types considered by wcslib. When `None`, only the standard image header keywords are considered (and the underlying wcspih() C function is called). To use binary table image array or pixel list keywords, *keysel* must be set.

  Each element in the list should be one of the following strings:
  - 'image': Image header keywords
  - 'binary': Binary table image array keywords
  - 'pixel': Pixel list keywords

  Keywords such as `EQUIna` or `RFRQna` that are common to binary table image arrays and pixel lists (including `WCSNna` and `TWCSna`) are selected by both 'binary' and 'pixel'.
- *colsel*: A sequence of table column numbers used to restrict the WCS transformations considered to only those pertaining to the specified columns. If `None`, there is no restriction.
- *fix* : When `True` (default), call `fix` on the resulting object to fix any non-standard uses in the header.

> **Warning:** pywcs supports arbitrary *n* dimensions for the core WCS (the transformations handled by WCSLIB). However, the Paper IV lookup table and SIP distortions must be two dimensional. Therefore, if you try to create a WCS object where the core WCS has a different number of dimensions than 2 and that object also contains a Paper IV lookup table or SIP distortion, a `ValueError` exception will be raised. To avoid this, consider using the *naxis* kwarg to select two dimensions from the core WCS.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `ValueError`: Invalid key.
- `KeyError`: Key not found in FITS header.
- `AssertionError`: Lookup table distortion present in the header but fobj not provided.

all_pix2sky(**args*, ***kwargs*)

Transforms pixel coordinates to sky coordinates by doing all of the following in order:

- Detector to image plane correction (optionally)
- SIP distortion correction (optionally)

- Paper IV table-lookup distortion correction (optionally)
- wcslib WCS transformation

Either two or three arguments may be provided.

- 2 arguments: An *N* x *naxis* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the sky coordinates, in degrees. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

An optional keyword argument, *ra_dec_order*, may be provided, that when `True` will ensure that sky coordinates are always given and returned in as (*ra*, *dec*) pairs, regardless of the order of the axes specified by the in the `CTYPE` keywords.

For a transformation that is not two-dimensional, the two-argument form must be used.

Note: The order of the axes for the result is determined by the `CTYPEia` keywords in the FITS header, therefore it may not always be of the form (*ra*, *dec*). The *lat*, *lng*, *lattyp* and *lngtyp* members can be used to determine the order of the axes.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `ValueError`: Invalid coordinate transformation parameters.
- `ValueError`: x- and y-coordinate arrays are not the same size.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

calcFootprint (*header=None*, *undistort=True*, *axes=None*, *center=True*)

Calculates the footprint of the image on the sky.

A footprint is defined as the positions of the corners of the image on the sky after all available distortions have been applied.

Parameters

header : astropy.io.fits header object, optional

undistort : bool, optional

If `True`, take SIP and distortion lookup table into account

axes : length 2 sequence ints, optional

If provided, use the given sequence as the shape of the image. Otherwise, use the `NAXIS1` and `NAXIS2` keywords from the header that was used to create this `WCS` object.

center : boolean

If True use the corner of the pixel, otherwise use the center.

Returns

corners : (4, 2) array of (*x*, *y*) coordinates.

The order is counter-clockwise starting with bottom left corner.

copy()

Return a shallow copy of the object.

Convenience method so user doesn't have to import the `copy` stdlib module.

deepcopy()

Return a deep copy of the object.

Convenience method so user doesn't have to import the `copy` stdlib module.

det2im(**args*, ***kwargs*)

Convert detector coordinates to image plane coordinates using Paper IV table-lookup distortion correction.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *2* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the pixel coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `ValueError`: Invalid coordinate transformation parameters.

footprint_to_file(*filename=None*, *color='green'*, *width=2*)

Writes out a ds9 style regions file. It can be loaded directly by ds9.

- *filename*: string. Output file name - default is `'footprint.reg'`
- *color*: string. Color to use when plotting the line.
- *width*: int. Width of the region line.

get_axis_types()

`list of dicts`

Similar to `self.wcsprm.axis_types` but provides the information in a more Python-friendly format.

Returns a list of dictionaries, one for each axis, each containing attributes about the type of that axis.

Each dictionary has the following keys:

- 'coordinate_type':
  - None: Non-specific coordinate type.
  - 'stokes': Stokes coordinate.
  - 'celestial': Celestial coordinate (including `CUBEFACE`).
  - 'spectral': Spectral coordinate.
- 'scale':
  - 'linear': Linear axis.

- 'quantized': Quantized axis (STOKES, CUBEFACE).
- 'non-linear celestial': Non-linear celestial axis.
- 'non-linear spectral': Non-linear spectral axis.
- 'logarithmic': Logarithmic axis.
- 'tabular': Tabular axis.

- 'group'
  - Group number, e.g. lookup table number
- 'number'
  - For celestial axes:
    - 0: Longitude coordinate.
    - 1: Latitude coordinate.
    - 2: CUBEFACE number.
  - For lookup tables:
    - the axis number in a multidimensional table.

CTYPEia in "4-3" form with unrecognized algorithm code will generate an error.

get_naxis(*header=None*)

p4_pix2foc(**args*, ***kwargs*)

Convert pixel coordinates to focal plane coordinates using Paper IV table-lookup distortion correction.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *2* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the focal coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

Exceptions:

- MemoryError: Memory allocation failed.
- ValueError: Invalid coordinate transformation parameters.

pix2foc(**args*, ***kwargs*)

Convert pixel coordinates to focal plane coordinates using the SIP polynomial distortion convention and Paper IV table-lookup distortion correction.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *2* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the focal coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `ValueError`: Invalid coordinate transformation parameters.

printwcs()
Temporary function for internal use.

rotateCD(*theta*)

sip_foc2pix(**args*, ***kwargs*)
Convert focal plane coordinates to pixel coordinates using the SIP polynomial distortion convention.

Paper IV table lookup distortion correction is not applied, even if that information existed in the FITS file that initialized this *WCS* object.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *2* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the pixel coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `ValueError`: Invalid coordinate transformation parameters.

sip_pix2foc(**args*, ***kwargs*)
Convert pixel coordinates to focal plane coordinates using the SIP polynomial distortion convention.

Paper IV table lookup distortion correction is not applied, even if that information existed in the FITS file that initialized this *WCS* object. To correct for that, use *pix2foc* or *p4_pix2foc*.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *2* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the focal coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `ValueError`: Invalid coordinate transformation parameters.

sub(*axes*)
Extracts the coordinate description for a subimage from a *WCS* object.

The world coordinate system of the subimage must be separable in the sense that the world coordinates at any point in the subimage must depend only on the pixel coordinates of the axes extracted. In practice, this means that the `PCi_ja` matrix of the original image must not contain non-zero off-diagonal terms that associate any of the subimage axes with any of the non-subimage axes.

`sub` can also add axes to a wcsprm struct. The new axes will be created using the defaults set by the Wcsprm constructor which produce a simple, unnamed, linear axis with world coordinates equal to the pixel coordinate. These default values can be changed before invoking `set`.

No checks are performed to verify that the coordinate axes are consistent, that is done by `set`.

- *axes*: int or a sequence.
  - If an int, include the first *N* axes in their original order.
  - If a sequence, may contain a combination of image axis numbers (1-relative) or special axis identifiers (see below). Order is significant; `axes[0]` is the axis number of the input image that corresponds to the first axis in the subimage, etc. Use an axis number of 0 to create a new axis using the defaults.
  - If `0`, `[]` or `None`, do a deep copy.

Coordinate axes types may be specified using either strings or special integer constants. The available types are:

- `'longitude'` / `WCSSUB_LONGITUDE`: Celestial longitude
- `'latitude'` / `WCSSUB_LATITUDE`: Celestial latitude
- `'cubeface'` / `WCSSUB_CUBEFACE`: Quadcube `CUBEFACE` axis
- `'spectral'` / `WCSSUB_SPECTRAL`: Spectral axis
- `'stokes'` / `WCSSUB_STOKES`: Stokes axis
- `'celestial'` / `WCSSUB_CELESTIAL`: An alias for the combination of `'longitude'`, `'latitude'` and `'cubeface'`.

Returns a *WCS* object, which is a deep copy of the original object.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `InvalidSubimageSpecificationError`: Invalid subimage specification (no spectral axis).
- `NonseparableSubimageCoordinateSystem`: Non-separable subimage coordinate system.

Note: Combinations of subimage axes of particular types may be extracted in the same order as they occur in the input image by combining the integer constants with the 'binary or' (`|`) operator. For example:

```
wcs.sub([WCSSUB_LONGITUDE | WCSSUB_LATITUDE | WCSSUB_SPECTRAL])
```

would extract the longitude, latitude, and spectral axes in the same order as the input image. If one of each were present, the resulting object would have three dimensions.

For convenience, `WCSSUB_CELESTIAL` is defined as the combination `WCSSUB_LONGITUDE | WCSSUB_LATITUDE | WCSSUB_CUBEFACE`.

The codes may also be negated to extract all but the types specified, for example:

```
wcs.sub([
  WCSSUB_LONGITUDE,
  WCSSUB_LATITUDE,
  WCSSUB_CUBEFACE,
  -(WCSSUB_SPECTRAL | WCSSUB_STOKES)])
```

The last of these specifies all axis types other than spectral or Stokes. Extraction is done in the order specified by `axes`, i.e. a longitude axis (if present) would be extracted first (via `axes[0]`) and not subsequently (via `axes[3]`). Likewise for the latitude and cubeface axes in this example.

The number of dimensions in the returned object may be less than or greater than the length of `axes`. However, it will never exceed the number of axes in the input image.

`to_fits` (*relax=False*, *wkey=None*)

Generate a `pyfits.HDUList` object with all of the information stored in this object. This should be logically identical to the input FITS file, but it will be normalized in a number of ways.

See `WCS.to_header` for some warnings about the output produced.

- *relax*: Degree of permissiveness:
  - `False`: Recognize only FITS keywords defined by the published WCS standard.
  - `True`: Admit all recognized informal extensions of the WCS standard.
  - `int`: a bit field selecting specific extensions to write. See *Header-writing relaxation constants* for details.

Returns a `pyfits.HDUList` object.

`to_header` (*relax=False*, *wkey=None*)

Generate a `pyfits.Header` object with the basic WCS and SIP information stored in this object. This should be logically identical to the input FITS file, but it will be normalized in a number of ways.

> **Warning:** This function does not write out Paper IV distortion information, since that requires multiple FITS header data units. To get a full representation of everything in this object, use `to_fits`.

The output header will almost certainly differ from the input in a number of respects:

1. The output header only contains WCS-related keywords. In particular, it does not contain syntactically-required keywords such as `SIMPLE`, `NAXIS`, `BITPIX`, or `END`.
2. Deprecated (e.g. `CROTAn`) or non-standard usage will be translated to standard (this is partially dependent on whether `fix` was applied).
3. Quantities will be converted to the units used internally, basically SI with the addition of degrees.
4. Floating-point quantities may be given to a different decimal precision.
5. Elements of the `PCi_j` matrix will be written if and only if they differ from the unit matrix. Thus, if the matrix is unity then no elements will be written.
6. Additional keywords such as `WCSAXES`, `CUNITia`, `LONPOLEa` and `LATPOLEa` may appear.
7. The original keycomments will be lost, although `to_header` tries hard to write meaningful comments.
8. Keyword order may be changed.

- *relax*: Degree of permissiveness:
  - `False`: Recognize only FITS keywords defined by the published WCS standard.
  - `True`: Admit all recognized informal extensions of the WCS standard.
  - `int`: a bit field selecting specific extensions to write. See *Header-writing relaxation constants* for details.

- *wkey*: A string. The name of a particular WCS transform to use. This may be either `' '` or `'A'-'Z'` and corresponds to the `"a"` part of the `CTYPEia` cards.

Returns a `pyfits.Header` object.

to_header_string(*relax=False*)
Identical to `to_header`, but returns a string containing the header cards.

wcs_pix2sky(**args*, ***kwargs*)
Transforms pixel coordinates to sky coordinates by doing only the basic wcslib transformation. No SIP or Paper IV table lookup distortion correction is applied. To perform distortion correction, see *all_pix2sky*, *sip_pix2foc*, *p4_pix2foc*, or *pix2foc*.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *naxis* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the sky coordinates, in degrees.. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

An optional keyword argument, *ra_dec_order*, may be provided, that when `True` will ensure that sky coordinates are always given and returned in as (*ra*, *dec*) pairs, regardless of the order of the axes specified by the in the `CTYPE` keywords.

For a transformation that is not two-dimensional, the two-argument form must be used.

Note: The order of the axes for the result is determined by the `CTYPEia` keywords in the FITS header, therefore it may not always be of the form (*ra*, *dec*). The *lat*, *lng*, *lattyp* and *lngtyp* members can be used to determine the order of the axes.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `ValueError`: Invalid coordinate transformation parameters.
- `ValueError`: x- and y-coordinate arrays are not the same size.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

wcs_sky2pix(**args*, ***kwargs*)
Transforms sky coordinates to pixel coordinates, using only the basic wcslib WCS transformation. No SIP or Paper IV table lookup distortion is applied.

Either two or three arguments may be provided.

- 2 arguments: An *N* x *naxis* array of *x*- and *y*-coordinates, and an *origin*.
- 3 arguments: 2 one-dimensional arrays of *x* and *y* coordinates, and an *origin*.

Here, *origin* is the coordinate in the upper left corner of the image. In FITS and Fortran standards, this is 1. In Numpy and C standards this is 0.

Returns the pixel coordinates. If the input was a single array and origin, a single array is returned, otherwise a tuple of arrays is returned.

An optional keyword argument, *ra_dec_order*, may be provided, that when `True` will ensure that sky coordinates are always given and returned in as (*ra*, *dec*) pairs, regardless of the order of the axes specified by the in the `CTYPE` keywords.

For a transformation that is not two-dimensional, the two-argument form must be used.

Note: The order of the axes for the input sky array is determined by the `CTYPEia` keywords in the FITS header, therefore it may not always be of the form (*ra*, *dec*). The *lat*, *lng*, *lattyp* and *lngtyp* members can be used to determine the order of the axes.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

cpdis1
DistortionLookupTable

The pre-linear transformation distortion lookup table, `CPDIS1`.

cpdis2
DistortionLookupTable

The pre-linear transformation distortion lookup table, `CPDIS2`.

det2im1
A *DistortionLookupTable* object for detector to image plane correction in the *x*-axis. A *DistortionLookupTable* object for detector to image plane correction in the *y*-axis. `int array[ndim]` (read-only)

The dimensions of the tabular array `data`.

det2im2
A *DistortionLookupTable* object for detector to image plane correction in the *y*-axis. `int array[ndim]` (read-only)

The dimensions of the tabular array `data`.

sip
Get/set the *Sip* object for performing SIP distortion correction.

wcs
A *Wcsprm* object to perform the basic wcslib WCS tranformation.

2.2.2 functions

pywcs.**find_all_wcs**(*header*, *relax=False*, *keysel=None*)
Find all the WCS transformations in the given header.

•*header*: A string or PyFITS header object.

•*relax*: Degree of permissiveness:

–`False`: Recognize only FITS keywords defined by the published WCS standard.

–`True`: Admit all recognized informal extensions of the WCS standard.

–`int`: a bit field selecting specific extensions to accept. See *Header-reading relaxation constants* for details.

•*keysel*: A list of flags used to select the keyword types considered by wcslib. When `None`, only the standard image header keywords are considered (and the underlying wcspih() C function is called). To use binary table image array or pixel list keywords, *keysel* must be set.

Each element in the list should be one of the following strings:

–'image': Image header keywords

–'binary': Binary table image array keywords

–'pixel': Pixel list keywords

Keywords such as `EQUIna` or `RFRQna` that are common to binary table image arrays and pixel lists (including `WCSNna` and `TWCSna`) are selected by both 'binary' and 'pixel'.

Returns a list of `WCS` objects.

2.2.3 Wcsprm

class pywcs.**Wcsprm**

celfix()
Translates AIPS-convention celestial projection types, `-NCP` and `-GLS`.

Returns `0` for success; `-1` if no change required.

cylfix()
Fixes WCS keyvalues for malformed cylindrical projections.

Returns `0` for success; `-1` if no change required.

datfix()
Translates the old `DATE-OBS` date format to year-2000 standard form `(yyyy-mm-ddThh:mm:ss)` and derives `MJD-OBS` from it if not already set. Alternatively, if *mjdobs* is set and *dateobs* isn't, then *datfix* derives *dateobs* from it. If both are set but disagree by more than half a day then `ValueError` is raised.

Returns `0` for success; `-1` if no change required.

fix(*translate_units=''*, *naxis=0*)
Applies all of the corrections handled separately by *datfix*, *unitfix*, *celfix*, *spcfix* and *cylfix*.

•*translate_units*: string. Do potentially unsafe translations of non-standard unit strings.

Although "S" is commonly used to represent seconds, its translation to "s" is potentially unsafe since the standard recognizes "S" formally as Siemens, however rarely that may be used. The same applies to "H" for hours (Henry), and "D" for days (Debye).

This string controls what to do in such cases, and is case-insensitive.

–If the string contains "s", translate "S" to "s".

–If the string contains "h", translate "H" to "h".

–If the string contains "d", translate "D" to "d".

Thus '' doesn't do any unsafe translations, whereas 'shd' does all of them.

•*naxis*: int array[naxis]. Image axis lengths. If this array is set to zero or None, then *cylfix* will not be invoked.

Returns a dictionary containing the following keys, each referring to a status string for each of the sub-fix functions that were called:

•*datfix*

•*unitfix*

•*celfix*

•*spcfix*

•*cylfix*

get_cdelt() → double array[naxis]
Coordinate increments (CDELTia) for each coord axis.

Returns the CDELT offsets in read-only form. Unlike the *cdelt* property, this works even when the header specifies the linear transformation matrix in one of the alternative CDi_ja or CROTAia forms. This is useful when you want access to the linear transformation matrix, but don't care how it was specified in the header.

get_pc() → double array[naxis][naxis]
Returns the PC matrix in read-only form. Unlike the *pc* property, this works even when the header specifies the linear transformation matrix in one of the alternative CDi_ja or CROTAia forms. This is useful when you want access to the linear transformation matrix, but don't care how it was specified in the header.

get_ps() → list of tuples
Returns PSi_ma keywords for each *i* and *m*. Returned as a list of tuples of the form (*i*, *m*, *value*):

•*i*: int. Axis number, as in PSi_ma, (i.e. 1-relative)

•*m*: int. Parameter number, as in PSi_ma, (i.e. 0-relative)

•*value*: string. Parameter value.

See also:

set_ps

get_pv() → list of tuples
Returns PVi_ma keywords for each *i* and *m*. Returned as a list of tuples of the form (*i*, *m*, *value*):

•*i*: int. Axis number, as in PVi_ma, (i.e. 1-relative)

•*m*: int. Parameter number, as in PVi_ma, (i.e. 0-relative)

•*value*: string. Parameter value.

Note that, if they were not given, *set* resets the entries for PVi_1a, PVi_2a, PVi_3a, and PVi_4a for longitude axis *i* to match (phi_0, theta_0), the native longitude and latitude of the reference point given by LONPOLEa and LATPOLEa.

See also:

set_pv

has_cd() → bool

Returns True if CDi_ja is present. CDi_ja is an alternate specification of the linear transformation matrix, maintained for historical compatibility.

Matrix elements in the IRAF convention are equivalent to the product CDi_ja = CDELTia * PCi_ja, but the defaults differ from that of the PCi_ja matrix. If one or more CDi_ja keywords are present then all unspecified CDi_ja default to zero. If no CDi_ja (or CROTAia) keywords are present, then the header is assumed to be in PCi_ja form whether or not any PCi_ja keywords are present since this results in an interpretation of CDELTia consistent with the original FITS specification.

While CDi_ja may not formally co-exist with PCi_ja, it may co-exist with CDELTia and CROTAia which are to be ignored.

See also:

cd

has_cdi_ja() → bool

Alias for *has_cd*. Maintained for backward compatibility. has_crota() -> bool

Returns True if CROTAia is present. CROTAia is an alternate specification of the linear transformation matrix, maintained for historical compatibility.

In the AIPS convention, CROTAia may only be associated with the latitude axis of a celestial axis pair. It specifies a rotation in the image plane that is applied *after* the CDELTia; any other CROTAia keywords are ignored.

CROTAia may not formally co-exist with PCi_ja. CROTAia and CDELTia may formally co-exist with CDi_ja but if so are to be ignored.

See also:

crota

has_crota() → bool

Returns True if CROTAia is present. CROTAia is an alternate specification of the linear transformation matrix, maintained for historical compatibility.

In the AIPS convention, CROTAia may only be associated with the latitude axis of a celestial axis pair. It specifies a rotation in the image plane that is applied *after* the CDELTia; any other CROTAia keywords are ignored.

CROTAia may not formally co-exist with PCi_ja. CROTAia and CDELTia may formally co-exist with CDi_ja but if so are to be ignored.

See also:

crota

has_crotaia()

has_crota_ia() -> bool

Alias for *has_crota*. Maintained for backward compatibility.

has_pc() → bool
Returns `True` if `PCi_ja` is present. `PCi_ja` is the recommended way to specify the linear transformation matrix.

See also:

pc

has_pci_ja() → bool
Alias for *has_pc*. Maintained for backward compatibility. The name of the unit being converted from.

This value always uses standard unit names, even if the `UnitConverter` was initialized with a non-standard unit name.

is_unity() → bool
Returns `True` if the linear transformation matrix (*cd*) is unity.

mix(*mixpix*, *mixcel*, *vspan*, *vstep*, *viter*, *world*, *pixcrd*, *origin*)
Given either the celestial longitude or latitude plus an element of the pixel coordinate, solves for the remaining elements by iterating on the unknown celestial coordinate element using *s2p*.

- *mixpix*: int. Which element on the pixel coordinate is given.
- *mixcel*: int. Which element of the celestial coordinate is given. If mixcel* = `1`, celestial longitude is given in `world[self.lng]`, latitude returned in `world[self.lat]`. If *mixcel* = `2`, celestial latitude is given in `world[self.lat]`, longitude returned in `world[self.lng]`.
- *vspan*: pair of floats. Solution interval for the celestial coordinate, in degrees. The ordering of the two limits is irrelevant. Longitude ranges may be specified with any convenient normalization, for example `(-120,+120)` is the same as `(240,480)`, except that the solution will be returned with the same normalization, i.e. lie within the interval specified.
- *vstep*: float. Step size for solution search, in degrees. If `0`, a sensible, although perhaps non-optimal default will be used.
- *viter*: int. If a solution is not found then the step size will be halved and the search recommenced. *viter* controls how many times the step size is halved. The allowed range is 5 - 10.
- *world*: double array[naxis]. World coordinate elements. `world[self.lng]` and `world[self.lat]` are the celestial longitude and latitude, in degrees. Which is given and which returned depends on the value of *mixcel*. All other elements are given. The results will be written to this array in-place.
- *pixcrd*: double array[naxis]. Pixel coordinate. The element indicated by *mixpix* is given and the remaining elements will be written in-place.
- *origin*: int. Specifies the origin of pixel values. The Fortran and FITS standards use an origin of 1. Numpy and C use array indexing with origin at 0.

Returns dictionary with the following keys:

- *phi* (double array[naxis])
- *theta* (double array[naxis])
  - Longitude and latitude in the native coordinate system of the projection, in degrees.
- *imgcrd* (double array[naxis])
  - Image coordinate elements. `imgcrd[self.lng]` and `imgcrd[self.lat]` are the projected *x*- and *y*-coordinates, in decimal degrees.
- *world* (double array[naxis])
  - Another reference to the *world* argument passed in.

Exceptions:

- `MemoryError` Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError` Ill-conditioned coordinate transformation parameters.
- `InvalidCoordinateError`: Invalid world coordinate.
- `NoSolutionError`: No solution found in the specified interval.

See also:

lat, *lng*

Note: Initially, the specified solution interval is checked to see if it's a "crossing" interval. If it isn't, a search is made for a crossing solution by iterating on the unknown celestial coordinate starting at the upper limit of the solution interval and decrementing by the specified step size. A crossing is indicated if the trial value of the pixel coordinate steps through the value specified. If a crossing interval is found then the solution is determined by a modified form of "regula falsi" division of the crossing interval. If no crossing interval was found within the specified solution interval then a search is made for a "non-crossing" solution as may arise from a point of tangency. The process is complicated by having to make allowance for the discontinuities that occur in all map projections.

Once one solution has been determined others may be found by subsequent invocations of *mix* with suitably restricted solution intervals.

Note the circumstance that arises when the solution point lies at a native pole of a projection in which the pole is represented as a finite curve, for example the zenithals and conics. In such cases two or more valid solutions may exist but *mix* only ever returns one.

Because of its generality, *mix* is very compute-intensive. For compute-limited applications, more efficient special-case solvers could be written for simple projections, for example non-oblique cylindrical projections.

---

p2s (*pixcrd*, *origin*)
Converts pixel to sky coordinates.

- *pixcrd*: double array[ncoord][nelem]. Array of pixel coordinates.
- *origin*: int. Specifies the origin of pixel values. The Fortran and FITS standards use an origin of 1. Numpy and C use array indexing with origin at 0.

Returns a dictionary with the following keys:

- *imgcrd*: double array[ncoord][nelem]
  - Array of intermediate sky coordinates. For celestial axes, `imgcrd[][self.lng]` and `imgcrd[][self.lat]` are the projected *x*-, and *y*-coordinates, in pseudo degrees. For spectral axes, `imgcrd[][self.spec]` is the intermediate spectral coordinate, in SI units.
- *phi*: double array[ncoord]
- *theta*: double array[ncoord]
  - Longitude and latitude in the native coordinate system of the projection, in degrees.

- *world*: double array[ncoord][nelem]
  - Array of sky coordinates. For celestial axes, `world[][self.lng]` and `world[][self.lat]` are the celestial longitude and latitude, in degrees. For spectral axes, `world[][self.spec]` is the intermediate spectral coordinate, in SI units.
- *stat*: int array[ncoord]
  - Status return value for each coordinate. `0` for success, `1+` for invalid pixel coordinate.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `ValueError`: *x*- and *y*-coordinate arrays are not the same size.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

See also:

lat, *lng*

print_contents()

Print the contents of the *Wcsprm* object to stdout. Probably only useful for debugging purposes, and may be removed in the future.

To get a string of the contents, use `repr`.

s2p(*sky*, *origin*)

Transforms sky coordinates to pixel coordinates.

- *sky*: double array[ncoord][nelem]. Array of sky coordinates, in decimal degrees.
- *origin*: int. Specifies the origin of pixel values. The Fortran and FITS standards use an origin of 1. Numpy and C use array indexing with origin at 0.

Returns a dictionary with the following keys:

- *phi*: double array[ncoord]
- *theta*: double array[ncoord]
  - Longitude and latitude in the native coordinate system of the projection, in degrees.
- *imgcrd*: double array[ncoord][nelem]
  - Array of intermediate sky coordinates. For celestial axes, `imgcrd[][self.lng]` and `imgcrd[][self.lat]` are the projected *x*-, and *y*-coordinates, in pseudo "degrees". For quadcube projections with a `CUBEFACE` axis, the face number is also returned in `imgcrd[][self.cubeface]`. For spectral axes, `imgcrd[][self.spec]` is the intermediate spectral coordinate, in SI units.
- *pixcrd*: double array[ncoord][nelem]
  - Array of pixel coordinates. Pixel coordinates are zero-based.
- *stat*: int array[ncoord]
  - Status return value for each coordinate. `0` for success, `1+` for invalid pixel coordinate.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError` Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

See also:

lat, *lng*

set()

Sets up a WCS object for use according to information supplied within it.

Note that this routine need not be called directly; it will be invoked by *p2s* and *s2p* if necessary.

Some attributes that are based on other attributes (such as *lattyp* on *ctype*) may not be correct until after *set* is called.

set strips off trailing blanks in all string members.

set recognizes the `NCP` projection and converts it to the equivalent `SIN` projection and it also recognizes `GLS` as a synonym for `SFL`. It does alias translation for the AIPS spectral types (`FREQ-LSR`, `FELO-HEL`, etc.) but without changing the input header keywords.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `SingularMatrixError`: Linear transformation matrix is singular.
- `InconsistentAxisTypesError`: Inconsistent or unrecognized coordinate axis types.
- `ValueError`: Invalid parameter value.
- `InvalidTransformError`: Invalid coordinate transformation parameters.
- `InvalidTransformError`: Ill-conditioned coordinate transformation parameters.

set_ps(*list*)

Sets `PSi_ma` keywords for each *i* and *m*. The input must be a sequence of tuples of the form (*i*, *m*, *value*):

- *i*: int. Axis number, as in `PSi_ma`, (i.e. 1-relative)
- *m*: int. Parameter number, as in `PSi_ma`, (i.e. 0-relative)
- *value*: string. Parameter value.

See also:

get_ps

set_pv(*list*)

Sets `PVi_ma` keywords for each *i* and *m*. The input must be a sequence of tuples of the form (*i*, *m*, *value*):

- *i*: int. Axis number, as in `PVi_ma`, (i.e. 1-relative)
- *m*: int. Parameter number, as in `PVi_ma`, (i.e. 0-relative)
- *value*: float. Parameter value.

See also:

get_pv

spcfix() → int

Translates AIPS-convention spectral coordinate types. {FREQ, VELO, FELO}-{OBS, HEL, LSR} (e.g. FREQ-LSR, VELO-OBS, FELO-HEL)

Returns 0 for success; -1 if no change required.

sptr(*ctype*, *i=-1*)

Translates the spectral axis in a WCS object. For example, a FREQ axis may be translated into ZOPT-F2W and vice versa.

- *ctype*: string. Required spectral CTYPEia, maximum of 8 characters. The first four characters are required to be given and are never modified. The remaining four, the algorithm code, are completely determined by, and must be consistent with, the first four characters. Wildcarding may be used, i.e. if the final three characters are specified as "???", or if just the eighth character is specified as "?", the correct algorithm code will be substituted and returned.
- *i*: int. Index of the spectral axis (0-relative). If i < 0 (or not provided), it will be set to the first spectral axis identified from the CTYPE keyvalues in the FITS header.

Exceptions:

- MemoryError: Memory allocation failed.
- SingularMatrixError: Linear transformation matrix is singular.
- InconsistentAxisTypesError: Inconsistent or unrecognized coordinate axis types.
- ValueError: Invalid parameter value.
- InvalidTransformError: Invalid coordinate transformation parameters.
- InvalidTransformError: Ill-conditioned coordinate transformation parameters.
- InvalidSubimageSpecificationError: Invalid subimage specification (no spectral axis).

sub(*axes*)

Extracts the coordinate description for a subimage from a *WCS* object.

The world coordinate system of the subimage must be separable in the sense that the world coordinates at any point in the subimage must depend only on the pixel coordinates of the axes extracted. In practice, this means that the PCi_ja matrix of the original image must not contain non-zero off-diagonal terms that associate any of the subimage axes with any of the non-subimage axes.

sub can also add axes to a wcsprm struct. The new axes will be created using the defaults set by the Wcsprm constructor which produce a simple, unnamed, linear axis with world coordinates equal to the pixel coordinate. These default values can be changed before invoking set.

No checks are performed to verify that the coordinate axes are consistent, that is done by set.

- *axes*: int or a sequence.
  - If an int, include the first *N* axes in their original order.
  - If a sequence, may contain a combination of image axis numbers (1-relative) or special axis identifiers (see below). Order is significant; axes[0] is the axis number of the input image that corresponds to the first axis in the subimage, etc. Use an axis number of 0 to create a new axis using the defaults.
  - If 0, [] or None, do a deep copy.

Coordinate axes types may be specified using either strings or special integer constants. The available types are:

- `'longitude'` / `WCSSUB_LONGITUDE`: Celestial longitude
- `'latitude'` / `WCSSUB_LATITUDE`: Celestial latitude
- `'cubeface'` / `WCSSUB_CUBEFACE`: Quadcube `CUBEFACE` axis
- `'spectral'` / `WCSSUB_SPECTRAL`: Spectral axis
- `'stokes'` / `WCSSUB_STOKES`: Stokes axis
- `'celestial'` / `WCSSUB_CELESTIAL`: An alias for the combination of `'longitude'`, `'latitude'` and `'cubeface'`.

Returns a *WCS* object, which is a deep copy of the original object.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `InvalidSubimageSpecificationError`: Invalid subimage specification (no spectral axis).
- `NonseparableSubimageCoordinateSystem`: Non-separable subimage coordinate system.

Note: Combinations of subimage axes of particular types may be extracted in the same order as they occur in the input image by combining the integer constants with the 'binary or' (`|`) operator. For example:

```
wcs.sub([WCSSUB_LONGITUDE | WCSSUB_LATITUDE | WCSSUB_SPECTRAL])
```

would extract the longitude, latitude, and spectral axes in the same order as the input image. If one of each were present, the resulting object would have three dimensions.

For convenience, `WCSSUB_CELESTIAL` is defined as the combination `WCSSUB_LONGITUDE | WCSSUB_LATITUDE | WCSSUB_CUBEFACE`.

The codes may also be negated to extract all but the types specified, for example:

```
wcs.sub([
  WCSSUB_LONGITUDE,
  WCSSUB_LATITUDE,
  WCSSUB_CUBEFACE,
  -(WCSSUB_SPECTRAL | WCSSUB_STOKES)])
```

The last of these specifies all axis types other than spectral or Stokes. Extraction is done in the order specified by `axes`, i.e. a longitude axis (if present) would be extracted first (via `axes[0]`) and not subsequently (via `axes[3]`). Likewise for the latitude and cubeface axes in this example.

The number of dimensions in the returned object may be less than or greater than the length of `axes`. However, it will never exceed the number of axes in the input image.

`to_header`(*relax=False*)

`to_header` translates a WCS object into a FITS header.

- If the *colnum* member is non-zero then a binary table image array header will be produced.
- Otherwise, if the *colax* member is set non-zero then a pixel list header will be produced.
- Otherwise, a primary image or image extension header will be produced.

The output header will almost certainly differ from the input in a number of respects:

1. The output header only contains WCS-related keywords. In particular, it does not contain syntactically-required keywords such as `SIMPLE`, `NAXIS`, `BITPIX`, or `END`.
2. Deprecated (e.g. `CROTAn`) or non-standard usage will be translated to standard (this is partially dependent on whether `fix` was applied).
3. Quantities will be converted to the units used internally, basically SI with the addition of degrees.
4. Floating-point quantities may be given to a different decimal precision.
5. Elements of the `PCi_j` matrix will be written if and only if they differ from the unit matrix. Thus, if the matrix is unity then no elements will be written.
6. Additional keywords such as `WCSAXES`, `CUNITia`, `LONPOLEa` and `LATPOLEa` may appear.
7. The original keycomments will be lost, although *to_header* tries hard to write meaningful comments.
8. Keyword order may be changed.

Keywords can be translated between the image array, binary table, and pixel lists forms by manipulating the *colnum* or *colax* members of the `WCS` object.

- *relax*: Degree of permissiveness:
  - `False`: Recognize only FITS keywords defined by the published WCS standard.
  - `True`: Admit all recognized informal extensions of the WCS standard.
  - `int`: a bit field selecting specific extensions to write. See *Header-writing relaxation constants* for details.

Returns a raw FITS header as a string.

unitfix(*translate_units=''*)

Translates non-standard `CUNITia` keyvalues. For example, `DEG` -> `deg`, also stripping off unnecessary whitespace.

- *translate_units*: string. Do potentially unsafe translations of non-standard unit strings.

  Although `"S"` is commonly used to represent seconds, its recognizes `"S"` formally as Siemens, however rarely that may be translation to `"s"` is potentially unsafe since the standard used. The same applies to `"H"` for hours (Henry), and `"D"` for days (Debye).

  This string controls what to do in such cases, and is case-insensitive.

  - If the string contains `"s"`, translate `"S"` to `"s"`.
  - If the string contains `"h"`, translate `"H"` to `"h"`.
  - If the string contains `"d"`, translate `"D"` to `"d"`.

  Thus `''` doesn't do any unsafe translations, whereas `'shd'` does all of them.

  See *FITS unit specification* for more information.

Returns `0` for success; `-1` if no change required.

alt

`str`

Character code for alternate coordinate descriptions. For example, the `"a"` in keyword names such as `CTYPEia`. This is a space character for the primary coordinate description, or one of the 26 upper-case letters, A-Z.

axis_types
int array[naxis]

An array of four-digit type codes for each axis.

- First digit (i.e. 1000s):
  - 0: Non-specific coordinate type.
  - 1: Stokes coordinate.
  - 2: Celestial coordinate (including CUBEFACE).
  - 3: Spectral coordinate.
- Second digit (i.e. 100s):
  - 0: Linear axis.
  - 1: Quantized axis (STOKES, CUBEFACE).
  - 2: Non-linear celestial axis.
  - 3: Non-linear spectral axis.
  - 4: Logarithmic axis.
  - 5: Tabular axis.
- Third digit (i.e. 10s):
  - 0: Group number, e.g. lookup table number
- The fourth digit is used as a qualifier depending on the axis type.
  - For celestial axes:
    - 0: Longitude coordinate.
    - 1: Latitude coordinate.
    - 2: CUBEFACE number.
  - For lookup tables: the axis number in a multidimensional table.

CTYPEia in "4-3" form with unrecognized algorithm code will have its type set to -1 and generate an error.

cd
double array[naxis][naxis]

The CDi_ja linear transformation matrix.

For historical compatibility, three alternate specifications of the linear transforations are available in wcslib. The canonical PCi_ja with CDELTia, CDi_ja, and the deprecated CROTAia keywords. Although the latter may not formally co-exist with PCi_ja, the approach here is simply to ignore them if given in conjunction with PCi_ja.

has_pc, *has_cd* and *has_crota* can be used to determine which of these alternatives are present in the header.

These alternate specifications of the linear transformation matrix are translated immediately to PCi_ja by *set* and are nowhere visible to the lower-level routines. In particular, *set* resets *cdelt* to unity if CDi_ja is present (and no PCi_ja). If no CROTAia is associated with the latitude axis, set reverts to a unity PCi_ja matrix.

cdelt
double array[naxis]

Coordinate increments (CDELTia) for each coord axis.

If a CDi_ja linear transformation matrix is present, a warning is raised and *cdelt* is ignored. The CDi_ja matrix may be deleted by:

```
del wcs.wcs.cd
```

An undefined value is represented by NaN.

cel_offset
boolean

If True, an offset will be applied to (x, y) to force (x,y) = (0,0) at the fiducial point, (phi_0, theta_0). Default is False.

cname
list of strings

A list of the coordinate axis names, from CNAMEia.

colax
int array[naxis]

An array recording the column numbers for each axis in a pixel list.

colnum
int

Where the coordinate representation is associated with an image-array column in a FITS binary table, this property may be used to record the relevant column number.

It should be set to zero for an image header or pixel list.

crder
double array[naxis]

The random error in each coordinate axis, CRDERia.

An undefined value is represented by NaN.

crota
double array[naxis]

CROTAia keyvalues for each coordinate axis.

For historical compatibility, three alternate specifications of the linear transforations are available in wcslib. The canonical PCi_ja with CDELTia, CDi_ja, and the deprecated CROTAia keywords. Although the latter may not formally co-exist with PCi_ja, the approach here is simply to ignore them if given in conjunction with PCi_ja.

has_pc, *has_cd* and *has_crota* can be used to determine which of these alternatives are present in the header.

These alternate specifications of the linear transformation matrix are translated immediately to PCi_ja by *set* and are nowhere visible to the lower-level routines. In particular, *set* resets *cdelt* to unity if CDi_ja is present (and no PCi_ja). If no CROTAia is associated with the latitude axis, set reverts to a unity PCi_ja matrix.

crpix
double array[naxis]

Coordinate reference pixels (CRPIXja) for each pixel axis.

crval
double array[naxis]

Coordinate reference values (CRVALia) for each coordinate axis.

csyer
double array[naxis]

The systematic error in the coordinate value axes, CSYERia.

An undefined value is represented by NaN.

ctype
list of strings[naxis]

List of CTYPEia keyvalues.

The *ctype* keyword values must be in upper case and there must be zero or one pair of matched celestial axis types, and zero or one spectral axis.

cubeface
int

Index into the pixcrd (pixel coordinate) array for the CUBEFACE axis. This is used for quadcube projections where the cube faces are stored on a separate axis.

The quadcube projections (TSC, CSC, QSC) may be represented in FITS in either of two ways:

- The six faces may be laid out in one plane and numbered as follows:

```
          0

 4  3  2  1  4  3  2

          5
```

  Faces 2, 3 and 4 may appear on one side or the other (or both). The sky-to-pixel routines map faces 2, 3 and 4 to the left but the pixel-to-sky routines accept them on either side.

- The COBE convention in which the six faces are stored in a three-dimensional structure using a CUBEFACE axis indexed from 0 to 5 as above.

These routines support both methods; *set* determines which is being used by the presence or absence of a CUBEFACE axis in *ctype*. *p2s* and *s2p* translate the CUBEFACE axis representation to the single plane representation understood by the lower-level projection routines.

cunit
list of strings[naxis]

List of CUNITia keyvalues which define the units of measurement of the CRVALia, CDELTia and CDi_ja keywords.

As CUNITia is an optional header keyword, *cunit* may be left blank but otherwise is expected to contain a standard units specification as defined by WCS Paper I. *unitfix* is available to translate commonly used non-standard units specifications but this must be done as a separate step before invoking *set*.

For celestial axes, if *cunit* is not blank, *set* uses wcsunits to parse it and scale *cdelt*, *crval*, and *cd* to decimal degrees. It then resets *cunit* to "deg".

For spectral axes, if *cunit* is not blank, *set* uses wcsunits to parse it and scale *cdelt*, *crval*, and *cd* to SI units. It then resets *cunit* accordingly.

set ignores *cunit* for other coordinate types; *cunit* may be used to label coordinate values.

dateavg
string

Representative mid-point of the date of observation in ISO format, yyyy-mm-ddThh:mm:ss.

See also:

dateobs

dateobs
string

Start of the date of observation in ISO format, yyyy-mm-ddThh:mm:ss.

See also:

dateavg

equinox
double

The equinox associated with dynamical equatorial or ecliptic coordinate systems, EQUINOXa (or EPOCH in older headers). Not applicable to ICRS equatorial or ecliptic coordinates.

An undefined value is represented by NaN.

imgpix_matrix
double array[2][2] (read-only)

Inverse of the matrix containing the product of the CDELTia diagonal matrix and the PCi_ja matrix.

lat
int (read-only)

The index into the sky coordinate array containing latitude values.

latpole
double

The native latitude of the celestial pole, LATPOLEa (deg).

lattyp
string (read-only)

Celestial axis type for latitude, e.g. "RA", "DEC", "GLON", "GLAT", etc. extracted from 'RA–', 'DEC-', 'GLON', 'GLAT', etc. in the first four characters of CTYPEia but with trailing dashes removed.

lng
int (read-only)

The index into the sky coordinate array containing longitude values.

lngtyp
string (read-only)

Celestial axis type for longitude, e.g. "RA", "DEC", "GLON", "GLAT", etc. extracted from 'RA–', 'DEC-', 'GLON', 'GLAT', etc. in the first four characters of CTYPEia but with trailing dashes removed.

lonpole
double

The native longitude of the celestial pole, LONPOLEa (deg).

mjdavg
double

Modified Julian Date (MJD = JD - 2400000.5), MJD-AVG, corresponding to DATE-AVG.

An undefined value is represented by NaN.

See also:

mjdobs

mjdobs
double

Modified Julian Date (MJD = JD - 2400000.5), MJD-OBS, corresponding to DATE-OBS.

An undefined value is represented by NaN.

See also:

mjdavg

name
string

The name given to the coordinate representation WCSNAMEa.

naxis
int (read-only)

The number of axes (pixel and coordinate), given by the NAXIS or WCSAXESa keyvalues.

The number of coordinate axes is determined at parsing time, and can not be subsequently changed.

It is determined from the highest of the following:

1. NAXIS
2. WCSAXESa
3. The highest axis number in any parameterized WCS keyword. The keyvalue, as well as the keyword, must be syntactically valid otherwise it will not be considered.

If none of these keyword types is present, i.e. if the header only contains auxiliary WCS keywords for a particular coordinate representation, then no coordinate description is constructed for it.

This value may differ for different coordinate representations of the same image.

obsgeo
double array[3]

Location of the observer in a standard terrestrial reference frame, OBSGEO-X, OBSGEO-Y, OBSGEO-Z (in meters).

An undefined value is represented by NaN.

pc
double array[naxis][naxis]

The PCi_ja (pixel coordinate) transformation matrix. The order is:

```
[[PC1_1, PC1_2],
 [PC2_1, PC2_2]]
```

For historical compatibility, three alternate specifications of the linear transforations are available in wcslib. The canonical PCi_ja with CDELTia, CDi_ja, and the deprecated CROTAia keywords. Although the latter may not formally co-exist with PCi_ja, the approach here is simply to ignore them if given in conjunction with PCi_ja.

has_pc, *has_cd* and *has_crota* can be used to determine which of these alternatives are present in the header.

These alternate specifications of the linear transformation matrix are translated immediately to PCi_ja by *set* and are nowhere visible to the lower-level routines. In particular, *set* resets *cdelt* to unity if CDi_ja is present (and no PCi_ja). If no CROTAia is associated with the latitude axis, set reverts to a unity PCi_ja matrix.

phi0
double

The native latitude of the fiducial point, i.e. the point whose celestial coordinates are given in ref[1:2]. If undefined (NaN) the initialization routine, *set*, will set this to a projection-specific default.

See also:

theta0

piximg_matrix
double array[2][2] (read-only)

Matrix containing the product of the CDELTia diagonal matrix and the PCi_ja matrix. double

The exponent of the unit conversion.

radesys
string

The equatorial or ecliptic coordinate system type, RADESYSa.

restfrq
double

Rest frequency (Hz) from RESTFRQa.

An undefined value is represented by NaN.

restwav
double

Rest wavelength (m) from RESTWAVa.

An undefined value is represented by NaN.

spec
int (read-only)

The index containing the spectral axis values.

specsys
string

Spectral reference frame (standard of rest), SPECSYSa.

See also:

ssysobs, *velosys*.

ssysobs
string

The actual spectral reference frame in which there is no differential variation in the spectral coordinate across the field-of-view, SSYSOBSa.

See also:

specsys, *velosys*

ssyssrc
string

The spectral reference frame (standard of rest) in which the redshift was measured, SSYSSRCa.

tab
list of Tabprm

A list of tabular coordinate objects associated with this WCS.

theta0
double

The native longitude of the fiducial point, i.e. the point whose celestial coordinates are given in ref[1:2]. If undefined (NaN) the initialization routine, *set*, will set this to a projection-specific default.

See also:

phi0

velangl
double

The angle in degrees that should be used to decompose an observed velocity into radial and transverse components.

An undefined value is represented by NaN.

velosys
double

The relative radial velocity (m/s) between the observer and the selected standard of rest in the direction of the celestial reference coordinate, VELOSYSa.

An undefined value is represented by NaN.

See also:

specsys, *ssysobs*

zsource
double

The redshift, ZSOURCEa, of the source.

An undefined value is represented by NaN.

2.2.4 Tabprm

class pywcs._pywcs.**Tabprm**
A class to store the information related to tabular coordinates, i.e. coordinates that are defined via a lookup table.

This class can not be constructed directly from Python, but instead is returned from *tab*.

print_contents()
Print the contents of the *Tabprm* object to stdout. Probably only useful for debugging purposes, and may be removed in the future.

To get a string of the contents, use repr.

set()
Allocates memory for work arrays in the Tabprm class and sets up the class according to information supplied within it.

Note that this routine need not be called directly; it will be invoked by functions that need it.

Exceptions:

- `MemoryError`: Memory allocation failed.
- `InvalidTabularParameters`: Invalid tabular parameters.

K

`int array[M]` (read-only)

An array of length `M` whose elements record the lengths of the axes of the coordinate array and of each indexing vector.

M

`int` (read-only)

Number of tabular coordinate axes.

coord

`double array[K_M]...[K_2][K_1][M]`

The tabular coordinate array, with the dimensions:

```
(K_M, ... K_2, K_1, M)
```

(see *K*) i.e. with the `M` dimension varying fastest so that the `M` elements of a coordinate vector are stored contiguously in memory.

crval

`double array[M]`

Array whose elements contain the index value for the reference pixel for each of the tabular coordinate axes.

delta

`double array[M]` (read-only)

Array of interpolated indices into the coordinate array such that Upsilon_m, as defined in Paper III, is equal to (*p0* [m] + 1) + delta[m].

extrema

`double array[K_M]...[K_2][2][M]` (read-only)

An array recording the minimum and maximum value of each element of the coordinate vector in each row of the coordinate array, with the dimensions:

```
(K_M, ... K_2, 2, M)
```

(see *K*). The minimum is recorded in the first element of the compressed K_1 dimension, then the maximum. This array is used by the inverse table lookup function to speed up table searches.

map

`int array[M]`

A vector of length *M* that defines the association between axis *m* in the *M*-dimensional coordinate array (1 <= *m* <= *M*) and the indices of the intermediate world coordinate and world coordinate arrays.

When the intermediate and world coordinate arrays contain the full complement of coordinate elements in image-order, as will usually be the case, then `map[m-1] == i-1` for axis *i* in the *N*-dimensional image (1 <= *i* <= *N*). In terms of the FITS keywords:

```
map[PVi_3a - 1] == i - 1.
```

However, a different association may result if the intermediate coordinates, for example, only contains a (relevant) subset of intermediate world coordinate elements. For example, if *M* == 1 for an image with *N* > 1, it is possible to fill the intermediate coordinates with the relevant coordinate element with `nelem` set to 1. In this case `map[0] = 0` regardless of the value of *i*.

nc
`int` (read-only)

Total number of coordinate vectors in the coordinate array being the product K_1 * K_2 * ... * K_M.

p0
`int array[M]`

Vector of length *M* of interpolated indices into the coordinate array such that Upsilon_m, as defined in Paper III, is equal to `(p0[m] + 1) + delta[m]`.

sense
`int array[M]`

A vector of length *M* whose elements indicate whether the corresponding indexing vector is monotonically increasing (+1), or decreasing (-1).

2.2.5 DistortionLookupTable

class `pywcs.`**DistortionLookupTable**

- *table*: 2-dimensional array for the distortion lookup table.
- *crpix*: the distortion array reference pixel (a 2-tuple)
- *crval*: is the image array pixel coordinate (a 2-tuple)
- *cdelt*: is the grid step size (a 2-tuple)

Represents a single lookup table for a Paper IV distortion transformation.

get_offset *(*x, y*) -> (*x, y*)*
Returns the offset from the distortion table for pixel point (*x, y*).

cdelt
`double array[naxis]`

Coordinate increments (`CDELTia`) for each coord axis.

If a `CDi_ja` linear transformation matrix is present, a warning is raised and *cdelt* is ignored. The `CDi_ja` matrix may be deleted by:

```
del wcs.wcs.cd
```

An undefined value is represented by NaN.

crpix
`double array[naxis]`

Coordinate reference pixels (`CRPIXja`) for each pixel axis.

crval
`double array[naxis]`

Coordinate reference values (`CRVALia`) for each coordinate axis.

data
float array

The array data for the *DistortionLookupTable*.

2.2.6 Sip

class pywcs.**Sip**

The *Sip* class performs polynomial distortion correction using the SIP convention in both directions.

Shupe, D. L., M. Moshir, J. Li, D. Makovoz and R. Narron. 2005. "The SIP Convention for Representing Distortion in FITS Image Headers." ADASS XIV.

- *a*: double array[m+1][m+1]. The A_i_j polynomial for pixel to focal plane transformation. Its size must be ($m + 1$, $m + 1$) where m = A_ORDER.
- *b*: double array[m+1][m+1]. The B_i_j polynomial for pixel to focal plane transformation. Its size must be ($m + 1$, $m + 1$) where m = B_ORDER.
- *ap*: double array[m+1][m+1]. The AP_i_j polynomial for pixel to focal plane transformation. Its size must be ($m + 1$, $m + 1$) where m = AP_ORDER.
- *bp*: double array[m+1][m+1]. The BP_i_j polynomial for pixel to focal plane transformation. Its size must be ($m + 1$, $m + 1$) where m = BP_ORDER.
- *crpix*: double array[2]. The reference pixel.

foc2pix()
sip_foc2pix(*foccrd, origin*) -> double array[ncoord][nelem]

Convert focal plane coordinates to pixel coordinates using the SIP polynomial distortion convention.

- *foccrd*: double array[ncoord][nelem]. Array of focal plane coordinates.
- *origin*: int. Specifies the origin of pixel values. The Fortran and FITS standards use an origin of 1. Numpy and C use array indexing with origin at 0.

Returns an array of pixel coordinates.

Exceptions:

- MemoryError: Memory allocation failed.
- ValueError: Invalid coordinate transformation parameters.

pix2foc()
sip_pix2foc(*pixcrd, origin*) -> double array[ncoord][nelem]

Convert pixel coordinates to focal plane coordinates using the SIP polynomial distortion convention.

- *pixcrd*: double array[ncoord][nelem]. Array of pixel coordinates.
- *origin*: int. Specifies the origin of pixel values. The Fortran and FITS standards use an origin of 1. Numpy and C use array indexing with origin at 0.

Returns an array of focal plane coordinates.

Exceptions:

- MemoryError: Memory allocation failed.
- ValueError: Invalid coordinate transformation parameters.

a
double array[a_order+1][a_order+1]

The SIP A_i_j matrix used for pixel to focal plane transformation.

Its values may be changed in place, but it may not be resized, without creating a new *Sip* object.

a_order
int (read-only)

The order of the polynomial in the SIP A_i_j array (A_ORDER).

ap
double array[ap_order+1][ap_order+1]

The SIP AP_i_j matrix used for focal plane to pixel transformation. Its values may be changed in place, but it may not be resized, without creating a new *Sip* object.

ap_order
int (read-only)

The order of the polynomial in the SIP AP_i_j array (AP_ORDER). int array[naxis]

An array of four-digit type codes for each axis.

- First digit (i.e. 1000s):
  - 0: Non-specific coordinate type.
  - 1: Stokes coordinate.
  - 2: Celestial coordinate (including CUBEFACE).
  - 3: Spectral coordinate.
- Second digit (i.e. 100s):
  - 0: Linear axis.
  - 1: Quantized axis (STOKES, CUBEFACE).
  - 2: Non-linear celestial axis.
  - 3: Non-linear spectral axis.
  - 4: Logarithmic axis.
  - 5: Tabular axis.
- Third digit (i.e. 10s):
  - 0: Group number, e.g. lookup table number
- The fourth digit is used as a qualifier depending on the axis type.
  - For celestial axes:
    - 0: Longitude coordinate.
    - 1: Latitude coordinate.
    - 2: CUBEFACE number.
  - For lookup tables: the axis number in a multidimensional table.

CTYPEia in "4-3" form with unrecognized algorithm code will have its type set to -1 and generate an error.

b
double array[b_order+1][b_order+1]

The SIP B_i_j matrix used for pixel to focal plane transformation. Its values may be changed in place, but it may not be resized, without creating a new *Sip* object.

b_order
int (read-only)

The order of the polynomial in the SIP B_i_j array (B_ORDER).

bp
double array[bp_order+1][bp_order+1]

The SIP BP_i_j matrix used for focal plane to pixel transformation. Its values may be changed in place, but it may not be resized, without creating a new *Sip* object.

bp_order
int (read-only)

The order of the polynomial in the SIP BP_i_j array (BP_ORDER). double array[naxis][naxis]

The CDi_ja linear transformation matrix.

For historical compatibility, three alternate specifications of the linear transforations are available in wcslib. The canonical PCi_ja with CDELTia, CDi_ja, and the deprecated CROTAia keywords. Although the latter may not formally co-exist with PCi_ja, the approach here is simply to ignore them if given in conjunction with PCi_ja.

has_pc, *has_cd* and *has_crota* can be used to determine which of these alternatives are present in the header.

These alternate specifications of the linear transformation matrix are translated immediately to PCi_ja by *set* and are nowhere visible to the lower-level routines. In particular, *set* resets *cdelt* to unity if CDi_ja is present (and no PCi_ja). If no CROTAia is associated with the latitude axis, set reverts to a unity PCi_ja matrix.

crpix
double array[naxis]

Coordinate reference pixels (CRPIXja) for each pixel axis.

2.2.7 UnitConverter

class pywcs.**UnitConverter**
Creates an object for performing conversion from one system of units to another.

- *have* string: *FITS unit specification* to convert from, with or without surrounding square brackets (for inline specifications); text following the closing bracket is ignored.
- *want* string: *FITS unit specification* to convert to, with or without surrounding square brackets (for inline specifications); text following the closing bracket is ignored.
- *ctrl* string (optional): Do potentially unsafe translations of non-standard unit strings.

  Although "S" is commonly used to represent seconds, its recognizes "S" formally as Siemens, however rarely that may be translation to "s" is potentially unsafe since the standard used. The same applies to "H" for hours (Henry), and "D" for days (Debye).

  This string controls what to do in such cases, and is case-insensitive.

  – If the string contains "s", translate "S" to "s".

–If the string contains "h", translate "H" to "h".

–If the string contains "d", translate "D" to "d".

Thus '' doesn't do any unsafe translations, whereas 'shd' does all of them.

See *FITS unit specification* for more information.

Use the returned object's *convert* method to convert values from *have* to *want*.

This function is permissive in accepting whitespace in all contexts in a units specification where it does not create ambiguity (e.g. not between a metric prefix and a basic unit string), including in strings like "log (m ** 2)" which is formally disallowed.

Exceptions:

- ValueError: Invalid numeric multiplier.
- SyntaxError: Dangling binary operator.
- SyntaxError: Invalid symbol in INITIAL context.
- SyntaxError: Function in invalid context.
- SyntaxError: Invalid symbol in EXPON context.
- SyntaxError: Unbalanced bracket.
- SyntaxError: Unbalanced parenthesis.
- SyntaxError: Consecutive binary operators.
- SyntaxError: Internal parser error.
- SyntaxError: Non-conformant unit specifications.
- SyntaxError: Non-conformant functions.
- ValueError: Potentially unsafe translation.

convert(*array*)
Perform the unit conversion on the elements of the given *array*, returning an array of the same shape.

have
The name of the unit being converted from.

This value always uses standard unit names, even if the UnitConverter was initialized with a non-standard unit name.

offset
double

The offset of the unit conversion.

power
double

The exponent of the unit conversion.

scale
double

The scaling factor for the unit conversion.

want
The name of the unit being converted to.

This value always uses standard unit names, even if the `UnitConverter` was initialized with a non-standard unit name.

2.2.8 FITS unit specification

Supported units

The following units are supported by the FITS standard:

SI base & supplementary units

Quantity	Unit String	Meaning
length	m	metre
mass	kg	kilogram
time	s	second of time
plane angle	rad	radian
solid angle	sr	steradian
temperature	K	kelvin
electric current	A	ampere
amount of substance	mol	mole
luminous intensity	cd	candela

IAU-recognized derived units

Quantity	Unit String	Meaning	Equivalence
frequency	Hz	hertz	s^{-1}
energy	J	joule	N m
power	W	watt	J s^{-1}
electric potential	V	volts	J C^{-1}
force	N	newton	kg m s^{-2}
pressure, stress	Pa	pascal	N m^{-2}
electric charge	C	coulomb	A s
electric resistance	ohm	ohm (Ω)	V A^{-1}
electric conductance	S	siemens	A V^{-1}
electric capacitance	F	farad	C V^{-1}
magnetic flux	Wb	weber	V s
magnetic flux density	T	tesla	Wb m^{-2}
inductance	H	henry	Wb A^{-1}
luminous flux	lm	lumen	cd sr
illuminance	lx	lux	lm m^{-2}

Additional units

Quantity	Unit String	Meaning	Equivalence
mass	u	unified atomic mass unit	1.6605387 x 10^{-27} kg
mass	solMass	solar mass	1.9891 x 10^{30} kg
plane angle	deg	degree of arc	1.745533 x 10^{-2} rad
plane angle	arcsec	second of arc	4.848137 x 10^{-6} rad
plane angle	arcmin	minute of arc	2.90888 x 10^{-4} rad
time	min	minute	
time	h	hour	
time	d	day	8.64 x 10^{4} s
time	yr	year (Julian)	3.15576 x10^{-7} s (365.25 d)

Continued on next page

Table 2.1 – continued from previous page

Quantity	Unit String	Meaning	Equivalence
energy	eV	electron volt	1.602177 x 10^{-19} J
energy	erg	erg	10^{-7} J
energy	Ry	Rydberg	13.605692 eV
length	angstrom	angstrom	10^{-10} m
length	AU	astronomical unit	1.49598 x 10^{11} m
length	lyr	light year	9.460530 x 10^{-15} m
length	pc	parsec	3.0857 x 10^{-16} m
length	solRad	solar radius	6.9599 x 10^{8} m
events	count	counts	
events	photon	photons	
flux density	Jy	jansky	10^{-16} W m^{-2} Hz^{-1}
flux density	mag	(stellar) magnitude	
flux density	Crab	'crab'	
flux density	beam	beam	Jy/beam
flux density	solLum	solar luminosity	
magnetic field	G	gauss	10^{-4} T
area	pixel	(image/detector) pixel	
area	voxel	3-d analog of pixel	
area	barn	barn	10^{-28} m^2
device	chan	(detector) channel	
device	byte	(computer) byte	
device	bit	(computer) bits	
device	adu	A/D converter units	
misc	bin	numerous applications	
misc	Sun	wrt. sun	

Potentially unsafe translations of `"D"`, `"H"`, and `"S"`, are optional, using the *translate_units* parameter.

Unit aliases

When converting non-standard units to standard ones, a case-sensitive match is required for the aliases listed below, in particular the only recognized aliases with metric prefixes are `"KM"`, `"KHZ"`, `"MHZ"`, and `"GHZ"`.

Unit	Recognized aliases
Angstrom	angstrom
arcmin	arcmins, ARCMIN, ARCMINS
arcsec	arcsecs, ARCSEC, ARCSECS
beam	BEAM
byte	Byte
count	ct
d	day, days, (D), DAY, DAYS
deg	degree, degrees, DEG, DEGREE, DEGREES
GHz	GHZ
h	hr, (H), HR
Hz	hz, HZ
kHz	KHZ
Jy	JY
K	kelvin, kelvins, Kelvin, Kelvins, KELVIN, KELVINS
km	KM
m	metre, meter, metres, meters, M, METRE, METER, METRES, METERS
min	MIN
MHz	MHZ
Ohm	ohm
Pa	pascal, pascals, Pascal, Pascals, PASCAL, PASCALS
photon	ph
pixel	pixels, PIXEL, PIXELS, pix
rad	radian, radians, RAD, RADIAN, RADIANS
s	sec, second, seconds, (S), SEC, SECOND, SECONDS
V	volt, volts, Volt, Volts, VOLT, VOLTS
yr	year, years, YR, YEAR, YEARS

The aliases `"angstrom"`, `"ohm"`, and `"Byte"` for (Angstrom, Ohm, and byte) are recognized by pywcs/wcslib itself as an unofficial extension of the standard, but they are converted to the standard form here.

Prefixes

The following metric prefixes are supported:

Prefix	String	Magnitude
yocto	y	10^{-24}
zepto	z	10^{-21}
atto	a	10^{-18}
femto	f	10^{-15}
pico	p	10^{-12}
nano	n	10^{-9}
micro	u	10^{-6}
milli	m	10^{-3}
centi	c	10^{-2}
deci	d	10^{-1}
deka	da	10^{1}
hecto	h	10^{2}
kilo	k	10^{3}
Mega	M	10^{6}
Giga	G	10^{9}
Tera	T	10^{12}
Peta	P	10^{15}
Exa	E	10^{18}
Zetta	Z	10^{21}
Yotta	Y	10^{24}

Table 6 of WCS Paper I lists eleven units for which metric prefixes are allowed. However, in this implementation only prefixes greater than unity are allowed for `"a"` (annum), `"yr"` (year), `"pc"` (parsec), `"bit"`, and `"byte"`, and only prefixes less than unity are allowed for `"mag"` (stellar magnitude).

Metric prefix `"P"` (peta) is specifically forbidden for `"a"` (annum) to avoid confusion with `"Pa"` (Pascal, not peta-annum). Note that metric prefixes are specifically disallowed for `"h"` (hour) and `"d"` (day) so that `"ph"` (photons) cannot be interpreted as pico-hours, nor `"cd"` (candela) as centi-days.

Operators

A compound unit is considered to be formed by a series of sub-strings of component units & mathematical operations. Each of these sub-strings must be separated by at least one space or a mathematical operator (`*` or `/`).

Multiplication

Multiplicative units can be specified either:

- by simply using one or more preceding spaces, e.g. `str1 str2` (The recommended method).
- by the use of a single asterisk (`*`) with optional whitespace, e.g. `str1 * str2`.

Division

Units which form the denominator of a compound expression can be specified either:

- by using a slash (`/`) with optional whitespace, e.g. `str1 / str2`. If such a syntax is used, it is recommended that no space is included between the slash and the unit string.
- by raising a multiplicative unit to a negative power (see below).

It should be stressed that the slash character only effects the sub-string it immediately precedes. Thus, unless brackets are used, subsequent sub-strings which also form part of the denominator of the compound expression must also be preceded by a slash. For example, `str1 /str2 str3` is equivalent to `str1 str3 /str2` whilst `str1 /str2 /str3` is equivalent to `str1 /(str2 * str3)`.

Raising to Powers

A unit string raised to the power *y* is specified:

- by using two asterisks (`**`) followed by the index enclosed within round brackets and with no preceding or intervening spaces, e.g. `str1**(y)` or `str1**(-y)`.

However, if *y* is positive, then the brackets need not be included, but a following space is recommended if additional sub-strings follow.

Use of brackets

Any number of pairs of round brackets (`()`) may be used within the string for a compound unit in order to prevent ambiguities. As described within this section, a number of rules always/often require their use. However, it is suggested that conservative use is made of such pairs of brackets in order to minimize the total length of compound strings. (It should be remembered that a maximum of 68 characters are allowed in the card image of keywords.)

Avoidance of underflows & overflows

The inclusion of numerical factors within the unit string should generally be avoided (by the use of multiples and/or submultiples of component basic units).

However, occasionally it may be preferable to include such factors on the grounds of user-friendliness and/or to minimize the risk of computer under- or overflows. In such cases, the numerical factor can simply be considered a basic unit string.

The following additional guidelines are suggested:

- the numerical factor should precede any unit strings
- only powers of 10 are used as numerical factors

Mathematical Operations & Functions

A number of mathematical operations are supported. It should be noted that the (round) brackets are mandatory in all cases in which they are included in the table.

String	Meaning
`str1*str2`	Multiplication
`str1 /str2`	Division
`str1**(y)`	Raised to the power *y*
`log(str1)`	Common Logarithm (to base 10)
`ln(str1)`	Natural Logarithm
`exp(str1)`	Exponential ($\exp^{str1}$)
`sqrt(str1)`	Square root
`sin(str1)`	Sine
`cos(str1)`	Cosine
`tan(str1)`	Tangent
`asin(str1)`	Arc Sine
`acos(str1)`	Arc Cosine
`atan(str1)`	Arc Tangent
`sinh(str1)`	Hyperbolic Sine
`cosh(str1)`	Hyperbolic Cosine
`tanh(str1)`	Hyperbolic Tangent

Function types `log()`, `ln()` and `exp()` may only occur at the start of the units specification.

2.2.9 Relax constants

Header-reading relaxation constants

WCS, *Wcsprm* and *find_all_wcs* have a *relax* argument, which may be either True, False or an int.

- If True, all non-standard WCS extensions recognized by the parser will be handled.
- If False (default), none of the extensions (even those in the errata) will be handled. Non-conformant keywords will be handled in the same way as non-WCS keywords in the header, i.e. by simply ignoring them.
- If an int, is is a bit field to provide fine-grained control over what non-standard WCS keywords to accept. The flag bits are subject to change in future and should be set by using the constants beginning with WCSHDR_ in the *pywcs* module.

  For example, to accept CD00i00j and PC00i00j use:

```
relax = pywcs.WCSHDR_CD00i00j | pywcs.WCSHDR_PC00i00j
```

  The parser always treats EPOCH as subordinate to EQUINOXa if both are present, and VSOURCEa is always subordinate to ZSOURCEa.

  Likewise, VELREF is subordinate to the formalism of WCS Paper III.

The flag bits are:

- WCSHDR_none: Don't accept any extensions (not even those in the errata). Treat non-conformant keywords in the same way as non-WCS keywords in the header, i.e. simply ignore them. (This is equivalent to the default behavior or passing False)
- WCSHDR_all: Accept all extensions recognized by the parser. (This is equivalent to passing True).
- WCSHDR_CROTAia: Accept CROTAia, iCROTna, TCROTna
- WCSHDR_EPOCHa: Accept EPOCHa.
- WCSHDR_VELREFa: Accept VELREFa.

  The constructor always recognizes the AIPS-convention keywords, CROTAn, EPOCH, and VELREF for the primary representation (a = ' ') but alternates are non-standard.

  The constructor accepts EPOCHa and VELREFa only if WCSHDR_AUXIMG is also enabled.

- WCSHDR_CD00i00j: Accept CD00i00j.
- WCSHDR_PC00i00j: Accept PC00i00j.
- WCSHDR_PROJPn: Accept PROJPn

  These appeared in early drafts of WCS Paper I+II (before they were split) and are equivalent to CDi_ja, PCi_ja, and PVi_ma for the primary representation (a = ' '). PROJPn is equivalent to PVi_ma with m = n <= 9, and is associated exclusively with the latitude axis.

- **WCSHDR_RADECSYS: Accept RADECSYS. This appeared in early** drafts of WCS Paper I+II and was subsequently replaced by RADESYSa. The construtor accepts RADECSYS only if WCSHDR_AUXIMG is also enabled.
- WCSHDR_VSOURCE: Accept VSOURCEa or VSOUna. This appeared in early drafts of WCS Paper III and was subsequently dropped in favour of ZSOURCEa and ZSOUna. The constructor accepts VSOURCEa only if WCSHDR_AUXIMG is also enabled.
- WCSHDR_DOBSn: Allow DOBSn, the column-specific analogue of DATE-OBS. By an oversight this was never formally defined in the standard.

- WCSHDR_LONGKEY: Accept long forms of the alternate binary table and pixel list WCS keywords, i.e. with "a" non- blank. Specifically:

```
jCRPXna  TCRPXna  :  jCRPXn  jCRPna  TCRPXn  TCRPna  CRPIXja
   -     TPCn_ka  :    -     ijPCna    -     TPn_ka  PCi_ja
   -     TCDn_ka  :    -     ijCDna    -     TCn_ka  CDi_ja
iCDLTna  TCDLTna  :  iCDLTn  iCDEna  TCDLTn  TCDEna  CDELTia
iCUNIna  TCUNIna  :  iCUNIn  iCUNna  TCUNIn  TCUNna  CUNITia
iCTYPna  TCTYPna  :  iCTYPn  iCTYna  TCTYPn  TCTYna  CTYPEia
iCRVLna  TCRVLna  :  iCRVLn  iCRVna  TCRVLn  TCRVna  CRVALia
iPVn_ma  TPVn_ma  :    -     iVn_ma    -     TVn_ma  PVi_ma
iPSn_ma  TPSn_ma  :    -     iSn_ma    -     TSn_ma  PSi_ma
```

  where the primary and standard alternate forms together with the image-header equivalent are shown rightwards of the colon.

  The long form of these keywords could be described as quasi- standard. TPCn_ka, iPVn_ma, and TPVn_ma appeared by mistake in the examples in WCS Paper II and subsequently these and also TCDn_ka, iPSn_ma and TPSn_ma were legitimized by the errata to the WCS papers.

  Strictly speaking, the other long forms are non-standard and in fact have never appeared in any draft of the WCS papers nor in the errata. However, as natural extensions of the primary form they are unlikely to be written with any other intention. Thus it should be safe to accept them provided, of course, that the resulting keyword does not exceed the 8-character limit.

  If WCSHDR_CNAMn is enabled then also accept:

```
iCNAMna  TCNAMna  :   ---   iCNAna    ---   TCNAna  CNAMEia
iCRDEna  TCRDEna  :   ---   iCRDna    ---   TCRDna  CRDERia
iCSYEna  TCSYEna  :   ---   iCSYna    ---   TCSYna  CSYERia
```

  Note that CNAMEia, CRDERia, CSYERia, and their variants are not used by pywcs but are stored as auxiliary information.

- WCSHDR_CNAMn: Accept iCNAMn, iCRDEn, iCSYEn, TCNAMn, TCRDEn, and TCSYEn, i.e. with a blank. While non-standard, these are the obvious analogues of iCTYPn, TCTYPn, etc.

- WCSHDR_AUXIMG: Allow the image-header form of an auxiliary WCS keyword with representation-wide scope to provide a default value for all images. This default may be overridden by the column-specific form of the keyword.

  For example, a keyword like EQUINOXa would apply to all image arrays in a binary table, or all pixel list columns with alternate representation a unless overridden by EQUIna.

  Specifically the keywords are:

```
LATPOLEa  for LATPna
LONPOLEa  for LONPna
RESTFREQ  for RFRQna
RESTFRQa  for RFRQna
RESTWAVa  for RWAVna
```

  whose keyvalues are actually used by WCSLIB, and also keywords that provide auxiliary information that is simply stored in the wcsprm struct:

```
EPOCH         -       ... (No column-specific form.)
EPOCHa        -       ... Only if WCSHDR_EPOCHa is set.
EQUINOXa  for EQUIna
RADESYSa  for RADEna
RADECSYS  for RADEna  ... Only if WCSHDR_RADECSYS is set.
SPECSYSa  for SPECna
```

```
    SSYSOBSa  for SOBSna
    SSYSSRCa  for SSRCna
    VELOSYSa  for VSYSna
    VELANGLa  for VANGna
    VELREF        -       ... (No column-specific form.)
    VELREFa       -       ... Only if WCSHDR_VELREFa is set.
    VSOURCEa  for VSOUna  ... Only if WCSHDR_VSOURCE is set.
    WCSNAMEa  for WCSNna  ... Or TWCSna (see below).
    ZSOURCEa  for ZSOUna

    DATE-AVG  for DAVGn
    DATE-OBS  for DOBSn
    MJD-AVG   for MJDAn
    MJD-OBS   for MJDOBn
    OBSGEO-X  for OBSGXn
    OBSGEO-Y  for OBSGYn
    OBSGEO-Z  for OBSGZn
```

where the image-header keywords on the left provide default values for the column specific keywords on the right.

Keywords in the last group, such as `MJD-OBS`, apply to all alternate representations, so `MJD-OBS` would provide a default value for all images in the header.

This auxiliary inheritance mechanism applies to binary table image arrays and pixel lists alike. Most of these keywords have no default value, the exceptions being `LONPOLEa` and `LATPOLEa`, and also `RADESYSa` and `EQUINOXa` which provide defaults for each other. Thus the only potential difficulty in using `WCSHDR_AUXIMG` is that of erroneously inheriting one of these four keywords.

Unlike `WCSHDR_ALLIMG`, the existence of one (or all) of these auxiliary WCS image header keywords will not by itself cause a *Wcsprm* object to be created for alternate representation `a`. This is because they do not provide sufficient information to create a non-trivial coordinate representation when used in conjunction with the default values of those keywords, such as `CTYPEia`, that are parameterized by axis number.

- `WCSHDR_ALLIMG`: Allow the image-header form of *all* image header WCS keywords to provide a default value for all image arrays in a binary table (n.b. not pixel list). This default may be overridden by the column-specific form of the keyword.

  For example, a keyword like `CRPIXja` would apply to all image arrays in a binary table with alternate representation `a` unless overridden by `jCRPna`.

  Specifically the keywords are those listed above for `WCSHDR_AUXIMG` plus:

```
    WCSAXESa  for WCAXna
```

  which defines the coordinate dimensionality, and the following keywords which are parameterized by axis number:

```
    CRPIXja   for jCRPna
    PCi_ja    for ijPCna
    CDi_ja    for ijCDna
    CDELTia   for iCDEna
    CROTAi    for iCROTn
    CROTAia       -       ... Only if WCSHDR_CROTAia is set.
    CUNITia   for iCUNna
    CTYPEia   for iCTYna
    CRVALia   for iCRVna
    PVi_ma    for iVn_ma
    PSi_ma    for iSn_ma
```

```
    CNAMEia   for iCNAna
    CRDERia   for iCRDna
    CSYERia   for iCSYna
```

where the image-header keywords on the left provide default values for the column specific keywords on the right.

This full inheritance mechanism only applies to binary table image arrays, not pixel lists, because in the latter case there is no well-defined association between coordinate axis number and column number.

Note that `CNAMEia`, `CRDERia`, `CSYERia`, and their variants are not used by pywcs but are stored in the *`Wcsprm`* object as auxiliary information.

Note especially that at least one *`Wcsprm`* object will be returned for each `a` found in one of the image header keywords listed above:

- If the image header keywords for `a` **are not** inherited by a binary table, then the struct will not be associated with any particular table column number and it is up to the user to provide an association.
- If the image header keywords for `a` **are** inherited by a binary table image array, then those keywords are considered to be "exhausted" and do not result in a separate *`Wcsprm`* object.

Header-writing relaxation constants

`to_header` and *`to_header_string`* has a *relax* argument which may be either `True`, `False` or an `int`.

- If `True`, write all recognized extensions.
- If `False` (default), none of the extensions (even those in the errata) will be written.
- If an `int`, is is a bit field to provide fine-grained control over what non-standard WCS keywords to accept. The flag bits are subject to change in future and should be set by using the constants beginning with `WCSHDO_` in the *`pywcs`* module.

The flag bits are:

- `WCSHDO_none`: Don't use any extensions.
- `WCSHDO_all`: Write all recognized extensions, equivalent to setting each flag bit.
- `WCSHDO_safe`: Write all extensions that are considered to be safe and recommended.
- `WCSHDO_DOBSn`: Write `DOBSn`, the column-specific analogue of `DATE-OBS` for use in binary tables and pixel lists. WCS Paper III introduced `DATE-AVG` and `DAVGn` but by an oversight `DOBSn` (the obvious analogy) was never formally defined by the standard. The alternative to using `DOBSn` is to write `DATE-OBS` which applies to the whole table. This usage is considered to be safe and is recommended.
- `WCSHDO_TPCn_ka`: WCS Paper I defined
  - `TPn_ka` and `TCn_ka` for pixel lists

    but WCS Paper II uses `TPCn_ka` in one example and subsequently the errata for the WCS papers legitimized the use of
  - `TPCn_ka` and `TCDn_ka` for pixel lists

    provided that the keyword does not exceed eight characters. This usage is considered to be safe and is recommended because of the non-mnemonic terseness of the shorter forms.
- `WCSHDO_PVn_ma`: WCS Paper I defined
  - `iVn_ma` and `iSn_ma` for bintables and

  - TVn_ma and TSn_ma for pixel lists

    but WCS Paper II uses iPVn_ma and TPVn_ma in the examples and subsequently the errata for the WCS papers legitimized the use of

  - iPVn_ma and iPSn_ma for bintables and
  - TPVn_ma and TPSn_ma for pixel lists

    provided that the keyword does not exceed eight characters. This usage is considered to be safe and is recommended because of the non-mnemonic terseness of the shorter forms.

- WCSHDO_CRPXna: For historical reasons WCS Paper I defined
  - jCRPXn, iCDLTn, iCUNIn, iCTYPn, and iCRVLn for bintables and
  - TCRPXn, TCDLTn, TCUNIn, TCTYPn, and TCRVLn for pixel lists

    for use without an alternate version specifier. However, because of the eight-character keyword constraint, in order to accommodate column numbers greater than 99 WCS Paper I also defined

  - jCRPna, iCDEna, iCUNna, iCTYna and iCRVna for bintables and
  - TCRPna, TCDEna, TCUNna, TCTYna and TCRVna for pixel lists

    for use with an alternate version specifier (the a). Like the PC, CD, PV, and PS keywords there is an obvious tendency to confuse these two forms for column numbers up to 99. It is very unlikely that any parser would reject keywords in the first set with a non-blank alternate version specifier so this usage is considered to be safe and is recommended.

- WCSHDO_CNAMna: WCS Papers I and III defined
  - iCNAna, iCRDna, and iCSYna for bintables and
  - TCNAna, TCRDna, and TCSYna for pixel lists

    By analogy with the above, the long forms would be

  - iCNAMna, iCRDEna, and iCSYEna for bintables and
  - TCNAMna, TCRDEna, and TCSYEna for pixel lists

    Note that these keywords provide auxiliary information only, none of them are needed to compute world coordinates. This usage is potentially unsafe and is not recommended at this time.

- WCSHDO_WCSNna: Write WCSNna instead of TWCSna for pixel lists. While the constructor treats WCSNna and TWCSna as equivalent, other parsers may not. Consequently, this usage is potentially unsafe and is not recommended at this time.
- WCSHDO_SIP: Write out Simnple Imaging Polynomial (SIP) keywords

CHAPTER

THREE

TESTING PYWCS

The unit tests are written for use with `nose`. To run them, install pywcs and then at the commandline:

```
nosetests pywcs.tests
```

p