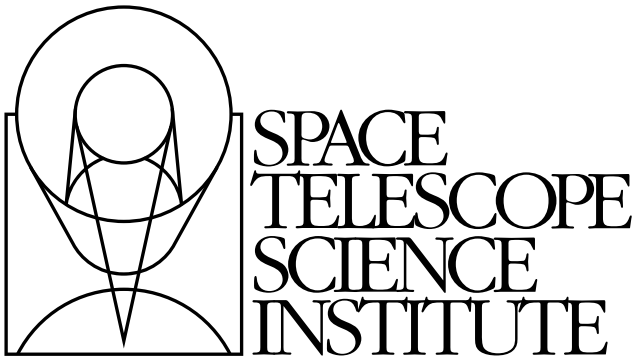

STWCS Documentation

Release 2011.xx

Nadezhda Dencheva, Warren Hack

April 18, 2016

This package provides support for WCS based distortion models and coordinate transformation. It relies on PyWCS (based on WCSLIB). It consists of two subpackages: UPDATEWCS and WCSUTIL. UPDATEWCS performs corrections to the basic WCS and includes other distortion infomation in the science files as header keywords or file extensions. WCSUTIL provides an HSTWCS object which extends pywcs.WCS object and provides HST instrument specific information as well as methods for coordinate tarnsformaiton. WCSUTIL also provides functions for manipulating alternate WCS descriptions in the headers.

Contents:

CHAPTER

ONE

WCSUTIL

This package provides an HSTWCS class which performs WCS based coordinate transformations and a module for managing alternate WCS's.

1.1 HSTWCS API

class `stwcs.wcsutil.hstwcs.`**`HSTWCS`**(*fobj=None*, *ext=None*, *minerr=0.0*, *wcskey=' '*)

Bases: `astropy.wcs.wcs.WCS`

Create a WCS object based on the instrument.

In addition to basic WCS keywords this class provides instrument specific information needed in distortion computation.

Parameters

fobj : str or `astropy.io.fits.HDUList` object or None

file name, e.g j9irw4b1q_flt.fits fully qualified filename[EXTNAME,EXTNUM], e.g. j9irw4b1q_flt.fits[sci,1] `astropy.io.fits` file object, e.g fits.open('j9irw4b1q_flt.fits'), in which case the user is responsible for closing the file object.

ext : int, tuple or None

extension number if ext is tuple, it must be ("EXTNAME", EXTNUM), e.g. ("SCI", 2) if ext is None, it is assumed the data is in the primary hdu

minerr : float

minimum value a distortion correction must have in order to be applied. If CPERRja, CQERRja are smaller than minerr, the corersponding distortion is not applied.

wcskey : str

A one character A-Z or " " used to retrieve and define an alternate WCS description.

`all_pix2world`(**args*, ***kwargs*)

Transforms pixel coordinates to world coordinates.

Performs all of the following in series:

- Detector to image plane correction (if present in the FITS file)
- **'SIP'_** distortion correction (if present in the FITS file)
- **'distortion paper'_** table-lookup correction (if present in the FITS file)
- **'wcslib'_** "core" WCS transformation

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *naxis* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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.

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

ra_dec_order : bool, optional

When `True` will ensure that world 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. Default is `False`.

Returns

result : array

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.

Raises

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.

Notes

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.

all_world2pix(**arg*, *accuracy=1.0e-4*, *maxiter=20*, *adaptive=False*, *detect_divergence=True*, *quiet=False*)

Performs full inverse transformation using iterative solution on full forward transformation with complete distortion model.

Parameters

accuracy : float, optional (Default = 1.0e-4)

Required accuracy of the solution. Iteration terminates when the correction to the solution found during the previous iteration is smaller (in the sence of the L2 norm) than `accuracy`.

maxiter : int, optional (Default = 20)

Maximum number of iterations allowed to reach the solution.

adaptive : bool, optional (Default = False)

Specifies whether to adaptively select only points that did not converge to a solution whithin the required accuracy for the next iteration. Default is recommended for HST as well as most other instruments.

Note: The *all_world2pix()* uses a vectorized implementation of the method of consecutive approximations (see `Notes` section below) in which it iterates over *all* input poits *regardless* until the required accuracy has been reached for *all* input points. In some cases it may be possible that *almost all* points have reached the required accuracy but there are only a few of input data points left for which additional iterations may be needed (this depends mostly on the characteristics of the geometric distortions for a given instrument). In this situation it may be advantageous to set `adaptive` = `True`in which case *all_world2pix()* will continue iterating *only* over the points that have not yet converged to the required accuracy. However, for the HST's ACS/WFC detector, which has the strongest distortions of all HST instruments, testing has shown that enabling this option would lead to a about 10-30% penalty in computational time (depending on specifics of the image, geometric distortions, and number of input points to be converted). Therefore, for HST instruments, it is recommended to set `adaptive` = `False`. The only danger in getting this setting wrong will be a performance penalty.

Note: When `detect_divergence` is `True`, *all_world2pix()* will automatically switch to the adaptive algorithm once divergence has been detected.

---

detect_divergence : bool, optional (Default = True)

Specifies whether to perform a more detailed analysis of the convergence to a solution. Normally *all_world2pix()* may not achieve the required accuracy if either the `tolerance` or `maxiter` arguments are too low. However, it may happen that for some geometric distortions the conditions of convergence for the the method of consecutive approximations used by *all_world2pix()* may not be satisfied, in which case consecutive approximations to the solution will diverge regardless of the `tolerance` or `maxiter` settings.

When `detect_divergence` is `False`, these divergent points will be detected as not having achieved the required accuracy (without further details). In addition, if `adaptive` is `False` then the algorithm will not know that the solution (for specific points) is diverging and will continue iterating and trying to "improve" diverging solutions. This may result in NaN or Inf values in the return results (in addition to a performance penalties). Even when `detect_divergence` is `False`,

all_world2pix(), at the end of the iterative process, will identify invalid results (NaN or Inf) as "diverging" solutions and will raise NoConvergence unless the quiet parameter is set to True.

When detect_divergence is True, *all_world2pix()* will detect points for which current correction to the coordinates is larger than the correction applied during the previous iteration **if** the requested accuracy **has not yet been achieved**. In this case, if adaptive is True, these points will be excluded from further iterations and if adaptive is False, *all_world2pix()* will automatically switch to the adaptive algorithm.

Note: When accuracy has been achieved, small increases in current corrections may be possible due to rounding errors (when adaptive is False) and such increases will be ignored.

Note: Setting detect_divergence to True will incurr about 5-10% performance penalty (in our testing on ACS/WFC images). Because the benefits of enabling this feature outweigh the small performance penalty, it is recommended to set detect_divergence to True, unless extensive testing of the distortion models for images from specific instruments show a good stability of the numerical method for a wide range of coordinates (even outside the image itself).

Note: Indices of the diverging inverse solutions will be reported in the divergent attribute of the raised NoConvergence object.

---

quiet : bool, optional (Default = False)

Do not throw NoConvergence exceptions when the method does not converge to a solution with the required accuracy within a specified number of maximum iterations set by maxiter parameter. Instead, simply return the found solution.

Raises

NoConvergence

The method does not converge to a solution with the required accuracy within a specified number of maximum iterations set by the maxiter parameter.

Notes

Inputs can either be (RA, Dec, origin) or (RADec, origin) where RA and Dec are 1-D arrays/lists of coordinates and RADec is an array/list of pairs of coordinates.

Using the method of consecutive approximations we iterate starting with the initial approximation, which is computed using the non-distorion-aware *wcs_world2pix()* (or equivalent).

The *all_world2pix()* function uses a vectorized implementation of the method of consecutive approximations and therefore it is highly efficient (>30x) when *all* data points that need to be converted from sky coordinates to image coordinates are passed at *once*. Therefore, it is advisable, whenever possible, to pass as input a long array of all points that need to be converted to *all_world2pix()* instead of calling *all_world2pix()* for each data point. Also see the note to the adaptive parameter.

Examples

```
>>> import stwcs
>>> from astropy.io import fits
>>> hdulist = fits.open('j94f05bgq_flt.fits')
>>> w = stwcs.wcsutil.HSTWCS(hdulist, ext=('sci',1))
>>> hdulist.close()
```

```
>>> ra, dec = w.all_pix2world([1,2,3],[1,1,1],1); print(ra); print(dec)
[ 5.52645241  5.52649277  5.52653313]
[-72.05171776 -72.05171295 -72.05170814]
>>> radec = w.all_pix2world([[1,1],[2,1],[3,1]],1); print(radec)
[[  5.52645241 -72.05171776]
 [  5.52649277 -72.05171295]
 [  5.52653313 -72.05170814]]
>>> x, y = w.all_world2pix(ra,dec,1)
>>> print(x)
[ 1.00000233  2.00000232  3.00000233]
>>> print(y)
[ 0.99999997  0.99999997  0.99999998]
>>> xy = w.all_world2pix(radec,1)
>>> print(xy)
[[ 1.00000233  0.99999997]
 [ 2.00000232  0.99999997]
 [ 3.00000233  0.99999998]]
>>> xy = w.all_world2pix(radec,1, maxiter=3, accuracy=1.0e-10, quiet=False)
NoConvergence: 'HSTWCS.all_world2pix' failed to converge to requested accuracy after 3 itera
```

```
>>>
Now try to use some diverging data:
>>> divradec = w.all_pix2world([[1.0,1.0],[10000.0,50000.0],[3.0,1.0]],1); print(divradec)
[[  5.52645241 -72.05171776]
 [  7.15979392 -70.81405561]
 [  5.52653313 -72.05170814]]
```

```
>>> try:
>>>   xy = w.all_world2pix(divradec,1, maxiter=20, accuracy=1.0e-4, adaptive=False, detect_d
>>> except stwcs.wcsutil.hstwcs.NoConvergence as e:
>>>   print("Indices of diverging points: {}".format(e.divergent))
>>>   print("Indices of poorly converging points: {}".format(e.failed2converge))
>>>   print("Best solution: {}".format(e.best_solution))
>>>   print("Achieved accuracy: {}".format(e.accuracy))
>>>   raise e
Indices of diverging points:
[1]
Indices of poorly converging points:
None
Best solution:
[[  1.00006219e+00   9.99999288e-01]
 [ -1.99440907e+06   1.44308548e+06]
 [  3.00006257e+00   9.99999316e-01]]
Achieved accuracy:
[[  5.98554253e-05   6.79918148e-07]
 [  8.59514088e+11   6.61703754e+11]
 [  6.02334592e-05   6.59713067e-07]]
Traceback (innermost last):
  File "<console>", line 8, in <module>
NoConvergence: 'HSTWCS.all_world2pix' failed to converge to the requested accuracy.
After 5 iterations, the solution is diverging at least for one input point.
```

```
>>> try:
>>>   xy = w.all_world2pix(divradec,1, maxiter=20, accuracy=1.0e-4, adaptive=False, detect_c
>>> except stwcs.wcsutil.hstwcs.NoConvergence as e:
>>>   print("Indices of diverging points: {}".format(e.divergent))
>>>   print("Indices of poorly converging points: {}".format(e.failed2converge))
>>>   print("Best solution: {}".format(e.best_solution))
>>>   print("Achieved accuracy: {}".format(e.accuracy))
>>>   raise e
Indices of diverging points:
[1]
Indices of poorly converging points:
None
Best solution:
[[  1.   1.]
 [ nan  nan]
 [  3.   1.]]
Achieved accuracy:
[[  0.   0.]
 [ nan  nan]
 [  0.   0.]]
Traceback (innermost last):
  File "<console>", line 8, in <module>
NoConvergence: 'HSTWCS.all_world2pix' failed to converge to the requested accuracy.
After 20 iterations, the solution is diverging at least for one input point.
```

calc_footprint(*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 : `Header` object, optional

Used to get `NAXIS1` and `NAXIS2` header and axes are mutually exclusive, alternative ways to provide the same information.

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 : bool, optional

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

Returns

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

The order is counter-clockwise starting with the 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*)

Convert detector coordinates to image plane coordinates using **'distortion paper'_** table-lookup correction.

The output is in absolute pixel coordinates, not relative to `CRPIX`.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *2* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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

result : array

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.

Raises

MemoryError

Memory allocation failed.

ValueError

Invalid coordinate transformation parameters.

dropaxis(*dropax*)

Remove an axis from the WCS.

Parameters

wcs : `WCS`

The WCS with naxis to be chopped to naxis-1

dropax : int

The index of the WCS to drop, counting from 0 (i.e., python convention, not FITS convention)

Returns

A new `WCS` instance with one axis fewer

fix(*translate_units=u''*, *naxis=None*)

Perform the fix operations from wcslib, and warn about any changes it has made.

Parameters

translate_units : str, optional

Specify which potentially unsafe translations of non-standard unit strings to perform. By default, performs none.

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], optional

Image axis lengths. If this array is set to zero or `None`, then `cylfix` will not be invoked.

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

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

Parameters

filename : str, optional

Output file name - default is `'footprint.reg'`

color : str, optional

Color to use when plotting the line.

width : int, optional

Width of the region line.

get_axis_types()

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

Returns

result : list of dicts

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.

p4_pix2foc(**args*)

Convert pixel coordinates to focal plane coordinates using **'distortion paper'_** table-lookup correction.

The output is in absolute pixel coordinates, not relative to `CRPIX`.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *2* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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

result : array

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.

Raises

MemoryError

Memory allocation failed.

ValueError

Invalid coordinate transformation parameters.

pc2cd()

pix2foc(**args*)

Convert pixel coordinates to focal plane coordinates using the **'SIP'_** polynomial distortion convention and **'distortion paper'_** table-lookup correction.

The output is in absolute pixel coordinates, not relative to `CRPIX`.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *2* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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

result : array

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.

Raises

MemoryError

Memory allocation failed.

ValueError

Invalid coordinate transformation parameters.

printwcs()

Print the basic WCS keywords.

readIDCCoeffs(*header*)

Reads in first order IDCTAB coefficients if present in the header

readModel(*update=False*, *header=None*)

Reads distortion model from IDCTAB.

If IDCTAB is not found ('N/A', "", or not found on disk), then if SIP coefficients and first order IDCTAB coefficients are present in the header, restore the idcmodel from the header. If not - assign None to self.idcmodel.

Parameters

header : `astropy.io.fits.Header`

fits extension header

update : bool (False)

if True - record the following IDCTAB quantities as header keywords: CX10, CX11, CY10, CY11, IDCSCALE, IDCTHETA, IDCXREF, IDCYREF, IDCV2REF, IDCV3REF

readModelFromIDCTAB(*header=None*, *update=False*)

Read distortion model from idc table.

Parameters

header : `astropy.io.fits.Header`

fits extension header

update : booln (False)

if True - save teh following as header keywords: CX10, CX11, CY10, CY11, IDCSCALE, IDCTHETA, IDCXREF, IDCYREF, IDCV2REF, IDCV3REF

reorient_celestial_first()

Reorient the WCS such that the celestial axes are first, followed by the spectral axis, followed by any others. Assumes at least celestial axes are present.

resetLTV()

Reset LTV values for polarizer data

The polarizer field is smaller than the detector field. The distortion coefficients are defined for the entire polarizer field and the LTV values are set as with subarray data. This may also be true for other special filters. This is a special case when the observation is considered a subarray in terms of detector field but

a full frame in terms of distortion model. To avoid shifting the distortion coefficients the LTV values are reset to 0.

rotateCD(*theta*)

setInstrSpecKw(*prim_hdr=None*, *ext_hdr=None*)
Populate the instrument specific attributes:

These can be in different headers but each instrument class has knowledge of where to look for them.

Parameters
prim_hdr : `astropy.io.fits.Header`

primary header

ext_hdr : `astropy.io.fits.Header`

extension header

setOrient()
Computes ORIENTAT from the CD matrix

setPscale()
Calculates the plate scale from the CD matrix

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

FITS WCS **'distortion paper'_** table lookup distortion correction is not applied, even if that information existed in the FITS file that initialized this `WCS` object.

Parameters
args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *2* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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
result : array

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.

Raises
MemoryError

Memory allocation failed.

ValueError

Invalid coordinate transformation parameters.

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

The output is in pixel coordinates, relative to `CRPIX`.

FITS WCS **'distortion paper'_** table lookup 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`.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *2* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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

result : array

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.

Raises

MemoryError

Memory allocation failed.

ValueError

Invalid coordinate transformation parameters.

`slice` (*view*, *numpy_order=True*)

Slice a WCS instance using a Numpy slice. The order of the slice should be reversed (as for the data) compared to the natural WCS order.

Parameters

view : tuple

A tuple containing the same number of slices as the WCS system. The `step` method, the third argument to a slice, is not presently supported.

numpy_order : bool

Use numpy order, i.e. slice the WCS so that an identical slice applied to a numpy array will slice the array and WCS in the same way. If set to `False`, the WCS will be sliced in FITS order, meaning the first slice will be applied to the *last* numpy index but the *first* WCS axis.

Returns

wcs_new : `WCS`

A new resampled WCS 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 object. 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`.

Parameters

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

new_wcs : `WCS` object

Raises

MemoryError

Memory allocation failed.

InvalidSubimageSpecificationError

Invalid subimage specification (no spectral axis).

NonseparableSubimageCoordinateSystem

Non-separable subimage coordinate system.

Notes

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.

swapaxes (*ax0*, *ax1*)

Swap axes in a WCS.

Parameters

wcs : `WCS`

The WCS to have its axes swapped

ax0 : int

ax1 : int

The indices of the WCS to be swapped, counting from 0 (i.e., python convention, not FITS convention)

Returns

A new `WCS` instance with the same number of axes, but two

swapped

to_fits (*relax=False*, *key=None*)

Generate an `astropy.io.fits.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 *to_header* for some warnings about the output produced.

Parameters

relax : bool or int, optional

Degree of permissiveness:

- `False` (default): Write all extensions that are considered to be safe and recommended.
- `True`: Write all recognized informal extensions of the WCS standard.
- `int`: a bit field selecting specific extensions to write. See relaxwrite for details.

key : str

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

hdulist : `astropy.io.fits.HDUList`

to_header (*relax=None*, *key=None*)

Generate an `astropy.io.fits.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 FITS WCS **'distortion paper'_** information, since that requires multiple FITS header data units. To get a full representation of everything in this object, use *to_fits*.

Parameters

relax : bool or int, optional

Degree of permissiveness:

- `False` (default): Write all extensions that are considered to be safe and recommended.
- `True`: Write all recognized informal extensions of the WCS standard.
- `int`: a bit field selecting specific extensions to write. See relaxwrite for details.

If the `relax` keyword argument is not given and any keywords were omitted from the output, an `AstropyWarning` is displayed. To override this, explicitly pass a value to `relax`.

key : str

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

header : `astropy.io.fits.Header`

Notes

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.

`to_header_string`(*relax=None*)

Identical to *`to_header`*, but returns a string containing the header cards.

`updatePscale`(*scale*)

Updates the CD matrix with a new plate scale

`wcs2header`(*sip2hdr=False*, *idc2hdr=True*, *wcskey=None*, *relax=False*)

Create a `astropy.io.fits.Header` object from WCS keywords.

If the original header had a CD matrix, return a CD matrix, otherwise return a PC matrix.

Parameters

sip2hdr : bool

If True - include SIP coefficients

`wcs_pix2world`(**args*, ***kwargs*)

Transforms pixel coordinates to world coordinates by doing only the basic **'wcslib'_** transformation.

No **'SIP'_** or **'distortion paper'_** table lookup correction is applied. To perform distortion correction, see `all_pix2world`, `sip_pix2foc`, `p4_pix2foc`, or `pix2foc`.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *naxis* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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.

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

ra_dec_order : bool, optional

When `True` will ensure that world 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. Default is `False`.

Returns

result : array

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

Raises

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.

Notes

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.

wcs_world2pix(**args*, ***kwargs*)

Transforms world coordinates to pixel coordinates, using only the basic **'wcslib'_** WCS transformation. No **'SIP'_** or **'distortion paper'_** table lookup transformation is applied.

Parameters

args : flexible

There are two accepted forms for the positional arguments:

- 2 arguments: An *N* x *naxis* array of coordinates, and an *origin*.
- more than 2 arguments: An array for each axis, followed by an *origin*. These arrays must be broadcastable to one another.

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.

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

ra_dec_order : bool, optional

When `True` will ensure that world 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. Default is `False`.

Returns

result : array

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.

Raises

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.

Notes

The order of the axes for the input world 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.

axis_type_names
World names for each coordinate axis

Returns
A list of names along each axis

celestial
A copy of the current WCS with only the celestial axes included

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.

det2im2
A `DistortionLookupTable` object for detector to image plane correction in the *y*-axis.

has_celestial

is_celestial

naxis1

naxis2

pixel_scale_matrix

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

wcs
A `Wcsprm` object to perform the basic **'wcslib'_** WCS transformation.

1.2 HSTWCS Examples

1.2.1 Create an HSTWCS Object

- Create an HSTWCS object using a pyfits HDUList and an extension number

```
fobj = pyfits.open('some_file.fits')
w = wcsutil.HSTWCS(fobj, 3)
```

- Create an HSTWCS object using a qualified file name.

```
w = wcsutil.HSTWCS('j9irw4b1q_flt.fits[sci,1]')
```

- Create an HSTWCS object using a file name and an extension number.

```
w = wcsutil.HSTWCS('j9irw4b1q_flt.fits', ext=2)
```

- Create an HSTWCS object from WCS with key 'O'.

```
w = wcsutil.HSTWCS('j9irw4b1q_flt.fits', ext=2, wcskey='O')
```

- Create a template HSTWCS object for a DEFAULT object.

```
w = wcsutil.HSTWCS(instrument='DEFAULT')
```

1.2.2 Coordinate Transformation Examples

All coordinate transformation functions accept input coordinates as 2D numpy arrays or 2 sequences of X and Y coordinates.

```
inpix = np.array([[1., 2.], [1,3], [1,4], [1,5]])
```

or

```
X = [1.,1.,1.,1.]
Y = np.array([2.,3.,4.,5.])
```

In addition all transformation functions require an `origin` parameter which specifies if the coordinates are 0 or 1 based. For example in FITS and Fortran, coordinates start from 1, while in Python and C, the index of the first image pixel is (0,0).

- Apply the entire detector to sky transformation at once:

```
outpix=w1.all_pix2sky(inpix,1)
outpix=w1.all_pix2sky(X, Y,1)
```

- The same transformation can be done in separate steps:

1. Apply the detector to image correction

```
dpx = w.det2im(inpix,1)
```

2. Aply the SIP polynomial distortion

```
spx = w.sip_pix2foc(dpx, 1)
```

3. Apply the non-polynomial distortion from the lookup table

```
lutpx = w.p4_pix2foc(dpx,1)
```

4. The undistorted coordinates are the sum of the input coordinates with the deltas for the distortion corrections.

```
fpix = dpx + (spx-dpx) +(lutpx-dpx)
```

5. Finally the transformation from undistorted to world coordinates is done by applying the linear WCS.

```
wpix = w.wcs_pix2sky(fpix, 1)
```

1.3 User Interface: altwcs

The functions in this module manage alternate WCS's in a header.

stwcs.wcsutil.altwcs.**archiveWCS** (*fname*, *ext*, *wcskey=' '*, *wcsname=' '*, *reusekey=False*)

Copy the primary WCS to the header as an alternate WCS with wcskey and name WCSNAME. It loops over all extensions in 'ext'

Parameters

fname : string or `astropy.io.fits.HDUList`

file name or a file object

ext : int, tuple, str, or list of integers or tuples (e.g.('sci',1))

fits extensions to work with If a string is provided, it should specify the EXTNAME of extensions with WCSs to be archived

wcskey : string "A"-"Z" or " "

if " ": get next available key if wcsname is also " " or try to get a key from WCSNAME value

wcsname : string

Name of alternate WCS description

reusekey : boolean

if True - overwrites a WCS with the same key

See also:

wcsutil.restoreWCS

Copy an alternate WCS to the primary WCS

Examples

Copy the primary WCS of an in memory headrlet object to an alternate WCS with key 'T'

```
>>> hlet=headerlet.createHeaderlet('junk.fits', 'hdr1.fits')
>>> altwcs.wcskeys(hlet[1].header)
['A']
>>> altwcs.archiveWCS(hlet, ext=[('SIPWCS',1),('SIPWCS',2)], wcskey='T')
>>> altwcs.wcskeys(hlet[1].header)
['A', 'T']
```

stwcs.wcsutil.altwcs.**restoreWCS** (*f*, *ext*, *wcskey=' '*, *wcsname=' '*)

Copy a WCS with key "WCSKEY" to the primary WCS

Reads in a WCS defined with wcskey and saves it as the primary WCS. Goes sequentially through the list of extensions in ext. Alternatively uses 'fromext' and 'toext'.

Parameters

f : str or `astropy.io.fits.HDUList`

file name or a file object

ext : int, tuple, str, or list of integers or tuples (e.g.('sci',1))

fits extensions to work with If a string is provided, it should specify the EXTNAME of extensions with WCSs to be archived

wcskey : str

"A"-"Z" - Used for one of 26 alternate WCS definitions. or " " - find a key from WCSNAMe value

wcsname : str

(optional) if given and wcskey is " ", will try to restore by WCSNAME value

See also:

archiveWCS, restore_from_to

stwcs.wcsutil.altwcs.**deleteWCS**(*fname*, *ext*, *wcskey=' '*, *wcsname=' '*)

Delete an alternate WCS defined with wcskey. If wcskey is " " try to get a key from WCSNAME.

Parameters

fname : str or a astropy.io.fits.HDUList

ext : int, tuple, str, or list of integers or tuples (e.g.('sci',1))

fits extensions to work with If a string is provided, it should specify the EXTNAME of extensions with WCSs to be archived

wcskey : str

one of 'A'-'Z' or " "

wcsname : str

Name of alternate WCS description

stwcs.wcsutil.altwcs.**wcsnames**(*fobj*, *ext=None*)

Returns a dictionary of wcskey: WCSNAME pairs

Parameters

fobj : stri, astropy.io.fits.HDUList or astropy.io.fits.Header

fits file name, fits file object or fits header

ext : int or None

extension number if None, fobj must be a header

stwcs.wcsutil.altwcs.**wcskeys**(*fobj*, *ext=None*)

Returns a list of characters used in the header for alternate WCS description with WCSNAME keyword

Parameters

fobj : str, astropy.io.fits.HDUList or astropy.io.fits.Header

fits file name, fits file object or fits header

ext : int or None

extension number if None, fobj must be a header

stwcs.wcsutil.altwcs.**available_wcskeys**(*fobj*, *ext=None*)

Returns a list of characters which are not used in the header with WCSNAME keyword. Any of them can be used to save a new WCS.

Parameters

fobj : str, astropy.io.fits.HDUList or astropy.io.fits.Header

fits file name, fits file object or fits header

ext : int or None

extension number if None, fobj must be a header

stwcs.wcsutil.altwcs.**next_wcskey**(*fobj*, *ext=None*)

Returns next available character to be used for an alternate WCS

Parameters

fobj : str, `astropy.io.fits.HDUList` or `astropy.io.fits.Header`

fits file name, fits file object or fits header

ext : int or None

extension number if None, fobj must be a header

stwcs.wcsutil.altwcs.**getKeyFromName**(*header*, *wcsname*)

If WCSNAME is found in header, return its key, else return None. This is used to update an alternate WCS repeatedly and not generate new keys every time.

Parameters

header : `astropy.io.fits.Header`

wcsname : str

value of WCSNAME

CHAPTER

TWO

UPDATEWCS

UPDATEWCS applies corrections to the WCS of an HST science file and adds reference information as header keywords and fits file extensions so that a science file contains all necessary information to represent astrometrically precise positions. The order in which the corrections are applied is important and is as follows:

- Detector to Image Correction
- Apply Time dependent distortion (if applciable)
- Recomputing the basic WCS
- Apply Velocity Aberration Correction
- Apply polynomial distortion through the SIP coefficients
- Apply non-polynomial distortion

Mathematically the entire transformation from detector to sky coordinates is described by:

$$(x', y') = DET2IM(x, y)$$

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} x' - CRPIX1 \\ y' - CRPIX2 \end{pmatrix}$$

$$\begin{pmatrix} \alpha \\ \delta \end{pmatrix} = \begin{pmatrix} CRVAL1 \\ CRVAL2 \end{pmatrix} + \begin{pmatrix} CD11 & CD12 \\ CD21 & CD22 \end{pmatrix} \begin{pmatrix} u' + f(u', v') + LT_x(x', y') \\ v' + g(u', v') + LT_y(x', y') \end{pmatrix}$$

where f(u',v') and g(u',v') represent the polynomial distortion correction specified as

$$f(u', v') = \sum_{p+q=2}^{AORDER} A_{pq} u'^p v'^q$$
$$g(u', v') = \sum_{p+q=2}^{BORDER} B_{pq} u'^p v'^q$$

where

- x', y' are the initial coordinates x,y with the 68th column correction applied through the DET2IM convention
- u',v' are the DET2IM-corrected coordinates relative to CRPIX1,CRPIX2
- LT_x, LT_y is the residual distortion in the lookup tables written to the header using the FITS Distortion Paper lookup table convention
- A, B are the SIP coefficients specified using the SIP convention

2.1 UPDATEWCS - User Interface

stwcs.updatewcs.**updatewcs**(*input*, *vacorr=True*, *tddcorr=True*, *npolcorr=True*, *d2imcorr=True*, *checkfiles=True*, *verbose=False*)

Updates HST science files with the best available calibration information. This allows users to retrieve from the archive self contained science files which do not require additional reference files.

Basic WCS keywords are updated in the process and new keywords (following WCS Paper IV and the SIP convention) as well as new extensions are added to the science files.

Parameters

input: a python list of file names or a string (wild card characters allowed)

input files may be in fits, geis or waiver fits format

vacorr: boolean

If True, vecocity aberration correction will be applied

tddcorr: boolean

If True, time dependent distortion correction will be applied

npolcorr: boolean

If True, a Lookup table distortion will be applied

d2imcorr: boolean

If True, detector to image correction will be applied

checkfiles: boolean

If True, the format of the input files will be checked, geis and waiver fits files will be converted to MEF format. Default value is True for standalone mode.

2.2 WCS Corrections

2.2.1 Time Dependent Distortion

class stwcs.updatewcs.corrections.**TDDCorr**

Apply time dependent distortion correction to distortion coefficients and basic WCS keywords. This correction **must** be done before any other WCS correction.

Parameters

ext_wcs: HSTWCS object

An HSTWCS object to be modified

ref_wcs: HSTWCS object

A reference HSTWCS object

Notes

Compute the ACS/WFC time dependent distortion terms as described in *[R1]* and apply the correction to the WCS of the observation.

The model coefficients are stored in the primary header of the IDCTAB. D_{ref} is the reference date. The computed corrections are saved in the science extension header as TDDALPHA and TDDBETA keywords.

$$TDDALPHA = A_0 + A_1 * (obsdate - D_{ref})$$

$$TDDBETA = B_0 + B_1 * (obsdate - D_{ref})$$

The time dependent distortion affects the IDCTAB coefficients, and the relative location of the two chips. Because the linear order IDCTAB coefficients ar eused in the computatuion of the NPOL extensions, the TDD correction affects all components of the distortion model.

Application of TDD to the IDCTAB polynomial coefficients: The TDD model is computed in Jay's frame, while the IDCTAB coefficients are in the HST V2/V3 frame. The coefficients are transformed to Jay's frame, TDD is applied and they are transformed back to the V2/V3 frame. This correction is performed in this class.

Application of TDD to the relative location of the two chips is done in `makewcs`.

References

[R1]

2.2.2 Velocity Aberration Correction

class `stwcs.updatewcs.corrections.`**`VACorr`**
Apply velocity aberation correction to WCS keywords.

Notes

Velocity Aberration is stored in the extension header keyword 'VAFACTOR'. The correction is applied to the CD matrix and CRVALs.

2.2.3 Simple Imaging Polynomial Coefficients

class `stwcs.updatewcs.corrections.`**`CompSIP`**
Compute Simple Imaging Polynomial (SIP) coefficients as defined in *[R4]* from IDC table coefficients.

This class transforms the TDD corrected IDCTAB coefficients into SIP format. It also applies a binning factor to the coefficients if the observation was binned.

References

[R4]

2.2.4 Non-Polynomial Distortion Correction

class `stwcs.updatewcs.npol.`**`NPOLCorr`**
Defines a Lookup table prior distortion correction as per WCS paper IV. It uses a reference file defined by the NPOLFILE (suffix 'NPL') keyword in the primary header.

Notes

- Using extensions in the reference file create a WCSDVARR extensions and add them to the science file.
- Add record-valued keywords to the science extension header to describe the lookup tables.
- Add a keyword 'NPOLEXT' to the science extension header to store the name of the reference file used to create the WCSDVARR extensions.

If WCSDVARR extensions exist and NPOLFILE is different from NPOLEXT, a subsequent update will overwrite the existing extensions. If WCSDVARR extensions were not found in the science file, they will be added.

It is assumed that the NPL reference files were created to work with IDC tables but will be applied with SIP coefficients. A transformation is applied to correct for the fact that the lookup tables will be applied before the first order coefficients which are in the CD matrix when the SIP convention is used.

2.2.5 Detector to Image Correction

class stwcs.updatewcs.det2im.**DET2IMCorr**

Defines a Lookup table prior distortion correction as per WCS paper IV. It uses a reference file defined by the D2IMFILE (suffix 'd2im') keyword in the primary header.

Notes

- Using extensions in the reference file create a WCSDVARR extensions and add them to the science file.
- Add record-valued keywords to the science extension header to describe the lookup tables.
- Add a keyword 'D2IMEXT' to the science extension header to store the name of the reference file used to create the WCSDVARR extensions.

If WCSDVARR extensions exist and D2IMFILE is different from D2IMEXT, a subsequent update will overwrite the existing extensions. If WCSDVARR extensions were not found in the science file, they will be added.

2.3 UPDATEWCS.UTILS functions

stwcs.updatewcs.utils.**build_sipname**(*fobj*, *fname=None*, *sipname=None*)

Build a SIPNAME from IDCTAB

Parameters

fobj : astropy.io.fits.HDUList

file object

fname : string

science file name (to be used if ROOTNAME is not present

sipname : string

user supplied SIPNAME keyword

Returns

sipname, idctab

stwcs.updatewcs.utils.**build_npolname**(*fobj*, *npolfile=None*)

Build a NPOLNAME from NPOLFILE

Parameters

fobj : astropy.io.fits.HDUList

file object

npolfile : string

user supplied NPOLFILE keyword

Returns

npolname, npolfile

stwcs.updatewcs.utils.**build_d2imname**(*fobj*, *d2imfile=None*)
Build a D2IMNAME from D2IMFILE

Parameters
fobj : astropy.io.fits.HDUList

file object

d2imfile : string

user supplied NPOLFILE keyword

Returns
d2imname, d2imfile

stwcs.updatewcs.utils.**build_distname**(*sipname*, *npolname*, *d2imname*)
Core function to build DISTNAME keyword value without the HSTWCS input.

CHAPTER

THREE

HEADERLET

The 'headerlet' serves as a mechanism for encapsulating WCS information for a single pointing so that it can be used to update the WCS solution of an image. The concept of a 'headerlet' seeks to provide a solution where only the WCS solution for an image that has been aligned to an astrometric catalog can be archived and retrieved for use in updating copies of that image's WCS information without getting the image data again. Multiple 'headerlets' could even be provided with each representing the alignment of an image to a different astrometric solution, giving the end user the option to get the solution that would allow them to best align their images with external data of interest to them. These benefits can only be realized with the proper definition of a 'headerlet' and the procedures used to define them and apply them to data.

The headerlet object needs to be as compact as possible while providing an unambigious and self-consistent WCS solution for an image while requiring a minimum level of software necessary to apply the headerlet to an image.

3.1 Headerlet File Structure

This new object complete with the NPOLFILE and the D2IMFILE extensions derived from the full FITS file fully describes the WCS of each chip and serves without further modification as the definition of the `headerlet`. The listing of the FITS extensions for a `headerlet` for the sample ACS/WFC exposure after writing it out to a file would then be:

```
EXT#  FITSNAME      FILENAME              EXTVE DIMENS       BITPI OBJECT

0     j8hw27c4q     j8hw27c4q_hdr.fits                       16
1       IMAGE       D2IMARR               1     4096         -32
2       IMAGE       WCSDVARR              1     64x32        -32
3       IMAGE       WCSDVARR              2     64x32        -32
4       IMAGE       WCSDVARR              3     64x32        -32
5       IMAGE       WCSDVARR              4     64x32        -32
6       IMAGE       SIPWCS                1                  8
7       IMAGE       SIPWCS                2                  8
```

This file now fully describes the WCS solution for this image, complete with all the distortion information used to originally define the solution. No further reference files or computations would be needed when this `headerlet` gets used to update an image.

The primary header must have 4 required keywords:

`HDRNAME` - a unique name for the headerlet

`DESTIM` - target image filename (the ROOTNAME keyword of the original archive filename)

`WCSNAME` - the value of WCSNAME<key> copied from the WCS which was used to create the headerlet

SIPNAME - the name of reference file which contained the original distortion model coefficients. A blank value or 'N/A' will indicate no SIP model was provided or applied. A value of 'UNKNOWN' indicates a SIP model of unknown origin.

NPOLFILE - the name of the NPOLFILE, the reference file which contained the original non-polynomial corrections. The same rules used for SIPNAME apply here as well.

D2IMFILE - the name of the D2IMFILE, the reference file which contained the detector to image correction (such as column width correction calibrations). The same rules used for SIPNAME apply here as well.

DISTNAME - a concatenation of SIPNAME, NPOLFILE, and D2IMFILE used as a quick reference for the distortion models included with this headerlet.

UPWCSVER - version of STWCS used to create the WCS of the original image

PYWCSVER - version of PyWCS used to create the WCS of the original image

3.2 User-Interface: headerlet

The headerlet module provides those functions necessary for creating, updating, and applying headerlets to FITS images.

3.2.1 Headerlet - User Interface

This module implements headerlets.

A headerlet serves as a mechanism for encapsulating WCS information which can be used to update the WCS solution of an image. The idea came up first from the desire for passing improved astrometric solutions for HST data and provide those solutions in a manner that would not require getting entirely new images from the archive when only the WCS information has been updated.

class stwcs.wcsutil.headerlet.**FuncNameLoggingFormatter**(*fmt=None*, *datefmt=None*)

format(*record*)

class stwcs.wcsutil.headerlet.**Headerlet**(*hdus=[]*, *file=None*, *logging=False*, *logmode='w'*)
A Headerlet class Ref: http://mediawiki.stsci.edu/mediawiki/index.php/Telescopedia:Headerlets

Parameters

hdus : list

List of HDUs to be used to create the headerlet object itself

file: string

File-like object from which HDUs should be read

logging: boolean

enable file logging

logmode: 'w' or 'a'

for internal use only, indicates whether the log file should be open in attach or write mode

apply_as_alternate(*fobj*, *attach=True*, *wcskey=None*, *wcsname=None*)
Copy this headerlet as an alternate WCS to fobj

Parameters

fobj: string, HDUList

science file/HDUList to which the headerlet should be applied

attach: boolean

flag indicating if the headerlet should be attached as a HeaderletHDU to fobj. If True checks that HDRNAME is unique in the fobj and stops if not.

wcskey: string

Key value (A-Z, except O) for this alternate WCS If None, the next available key will be used

wcsname: string

Name to be assigned to this alternate WCS WCSNAME is a required keyword in a Headerlet but this allows the user to change it as desired.

apply_as_primary (*fobj*, *attach=True*, *archive=True*, *force=False*)

Copy this headerlet as a primary WCS to fobj

Parameters

fobj: string, HDUList

science file to which the headerlet should be applied

attach: boolean

flag indicating if the headerlet should be attached as a HeaderletHDU to fobj. If True checks that HDRNAME is unique in the fobj and stops if not.

archive: boolean (default is True)

When the distortion model in the headerlet is the same as the distortion model of the science file, this flag indicates if the primary WCS should be saved as an alternate and a headerlet extension. When the distortion models do not match this flag indicates if the current primary and alternate WCSs should be archived as headerlet extensions and alternate WCS.

force: boolean (default is False)

When the distortion models of the headerlet and the primary do not match, and archive is False this flag forces an update of the primary

attach_to_file (*fobj*, *archive=False*)

Attach Headerlet as an HeaderletHDU to a science file

Parameters

fobj: string, HDUList

science file/HDUList to which the headerlet should be applied

archive: string

Specifies whether or not to update WCSCORR table when attaching

Notes

The algorithm used by this method: - verify headerlet can be applied to this file (based on DESTIM) - verify that HDRNAME is unique for this file - attach as HeaderletHDU to fobj

build_distname (*dest*)

Builds the DISTNAME for dest based on reference file names.

equal_distmodel(*dmodel*)

classmethod fromfile(*fileobj*, *mode='readonly'*, *memmap=False*, *save_backup=False*, *logging=False*, *logmode='w'*, ***kwargs*)

classmethod fromstring(*data*, ***kwargs*)

get_destination_model(*dest*)
Verifies that the headerlet can be applied to the observation

Determines whether or not the file specifies the same distortion model/reference files.

hverify()
Verify the headerlet file is a valid fits file and has the required Primary Header keywords

info(*columns=None*, *pad=2*, *maxwidth=None*, *output=None*, *clobber=True*, *quiet=False*)
Prints a summary of this headerlet The summary includes: HDRNAME WCSNAME DISTNAME SIPNAME NPOLFILE D2IMFILE

Parameters
columns: list

List of headerlet PRIMARY header keywords to report in summary By default (set to None), it will use the default set of keywords defined as the global list DEFAULT_SUMMARY_COLS

pad: int

Number of padding spaces to put between printed columns [Default: 2]

maxwidth: int

Maximum column width(not counting padding) for any column in summary By default (set to None), each column's full width will be used

output: string (optional)

Name of optional output file to record summary. This filename can contain environment variables. [Default: None]

clobber: bool

If True, will overwrite any previous output file of same name

quiet: bool

If True, will NOT report info to STDOUT

init_attrs()

summary(*columns=None*)
Returns a summary of this headerlet as a dictionary

The summary includes a summary of the distortion model as :
HDRNAME WCSNAME DISTNAME SIPNAME NPOLFILE D2IMFILE

Parameters
columns: list

List of headerlet PRIMARY header keywords to report in summary By default(set to None), it will use the default set of keywords defined as the global list DEFAULT_SUMMARY_COLS

Returns
summary: dict

Dictionary of values for summary

tofile (*fname*, *destim=None*, *hdrname=None*, *clobber=False*)
Write this headerlet to a file

Parameters
fname: string

file name

destim: string (optional)

provide a value for DESTIM keyword

hdrname: string (optional)

provide a value for HDRNAME keyword

clobber: boolean

a flag which allows to overwrte an existing file

verify_dest (*dest*, *fname*)
verifies that the headerlet can be applied to the observation

DESTIM in the primary header of the headerlet must match ROOTNAME of the science file (or the name of the destination file)

verify_hdrname (*dest*)
Verifies that the headerlet can be applied to the observation

Reports whether or not this file already has a headerlet with this HDRNAME.

class stwcs.wcsutil.headerlet.**HeaderletHDU**(*data=None*, *header=None*, *name=None*, ***kwargs*)
A non-standard extension HDU for encapsulating Headerlets in a file. These HDUs have an extension type of HDRLET and their EXTNAME is derived from the Headerlet's HDRNAME.

The data itself is a FITS file embedded within the HDU data. The file name is derived from the HDRNAME keyword, and should be in the form <HDRNAME>_hdr.fits. If the COMPRESS keyword evaluates to True, the tar file is compressed with gzip compression.

The structure of this HDU is the same as that proposed for the 'FITS' extension type proposed here: http://listmgr.cv.nrao.edu/pipermail/fitsbits/2002-April/thread.html

The Headerlet contained in the HDU's data can be accessed by the *headerlet* attribute.

classmethod fromheaderlet (*headerlet*, *compress=False*)
Creates a new HeaderletHDU from a given Headerlet object.

Parameters
headerlet : *Headerlet*

A valid Headerlet object.

compress : bool, optional

Gzip compress the headerlet data.

Returns
hlet : *HeaderletHDU*

A *HeaderletHDU* object for the given *Headerlet* that can be attached as an extension to an existing HDUList.

headerlet

Return the encapsulated headerlet as a Headerlet object.

This is similar to the hdulist property inherited from the FitsHDU class, though the hdulist property returns a normal HDUList object.

stwcs.wcsutil.headerlet.**apply_headerlet_as_alternate**(**args*, ***kw*)

Apply headerlet to a science observation as an alternate WCS

Parameters

filename: string or list of strings

File name(s) of science observation whose WCS solution will be updated

hdrlet: string or list of strings

Headerlet file(s), must match 1-to-1 with input filename(s)

attach: boolean

flag indicating if the headerlet should be attached as a HeaderletHDU to fobj. If True checks that HDRNAME is unique in the fobj and stops if not.

wcskey: string

Key value (A-Z, except O) for this alternate WCS If None, the next available key will be used

wcsname: string

Name to be assigned to this alternate WCS WCSNAME is a required keyword in a Headerlet but this allows the user to change it as desired.

logging: boolean

enable file logging

logmode: 'a' or 'w'

stwcs.wcsutil.headerlet.**apply_headerlet_as_primary**(**args*, ***kw*)

Apply headerlet 'hdrfile' to a science observation 'destfile' as the primary WCS

Parameters

filename: string or list of strings

File name(s) of science observation whose WCS solution will be updated

hdrlet: string or list of strings

Headerlet file(s), must match 1-to-1 with input filename(s)

attach: boolean

True (default): append headerlet to FITS file as a new extension.

archive: boolean

True (default): before updating, create a headerlet with the WCS old solution.

force: boolean

If True, this will cause the headerlet to replace the current PRIMARY WCS even if it has a different distortion model. [Default: False]

logging: boolean

enable file logging

logmode: 'w' or 'a'

log file open mode

stwcs.wcsutil.headerlet.**archive_as_headerlet**(**args*, ***kw*)

Save a WCS as a headerlet extension and write it out to a file.

This function will create a headerlet, attach it as an extension to the science image (if it has not already been archived) then, optionally, write out the headerlet to a separate headerlet file.

Either wcsname or wcskey must be provided, if both are given, they must match a valid WCS Updates wcscorr if necessary.

Parameters

filename: string or HDUList

Either a filename or PyFITS HDUList object for the input science file
An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

hdrname: string

Unique name for this headerlet, stored as HDRNAME keyword

sciext: string

name (EXTNAME) of extension that contains WCS to be saved

wcsname: string

name of WCS to be archived, if " ": stop

wcskey: one of A...Z or " " or "PRIMARY"

if " " or "PRIMARY" - archive the primary WCS

destim: string

DESTIM keyword if NOne, use ROOTNAME or science file name

sipname: string or None (default)

Name of unique file where the polynomial distortion coefficients were read from. If None, the behavior is: The code looks for a keyword 'SIPNAME' in the science header If not found, for HST it defaults to 'IDCTAB' If there is no SIP model the value is 'NOMODEL' If there is a SIP model but no SIPNAME, it is set to 'UNKNOWN'

npolfile: string or None (default)

Name of a unique file where the non-polynomial distortion was stored. If None: The code looks for 'NPOLFILE' in science header. If 'NPOLFILE' was not found and there is no npol model, it is set to 'NOMODEL' If npol model exists, it is set to 'UNKNOWN'

d2imfile: string

Name of a unique file where the detector to image correction was stored. If None: The code looks for 'D2IMFILE' in the science header. If 'D2IMFILE' is not found and there is no d2im correction, it is set to 'NOMODEL' If d2im correction exists, but 'D2IMFILE' is missing from science header, it is set to 'UNKNOWN'

author: string

Name of user who created the headerlet, added as 'AUTHOR' keyword to headerlet PRIMARY header

descrip: string

Short description of the solution provided by the headerlet This description will be added as the single 'DESCRIP' keyword to the headerlet PRIMARY header

history: filename, string or list of strings

Long (possibly multi-line) description of the solution provided by the headerlet. These comments will be added as 'HISTORY' cards to the headerlet PRIMARY header If filename is specified, it will format and attach all text from that file as the history.

logging: boolean

enable file folling

logmode: 'w' or 'a'

log file open mode

stwcs.wcsutil.headerlet.**attach_headerlet**(**args*, ***kw*)

Attach Headerlet as an HeaderletHDU to a science file

Parameters

filename: HDUList or list of HDULists

science file(s) to which the headerlet should be applied

hdrlet: string, Headerlet object or list of strings or Headerlet objects

string representing a headerlet file(s), must match 1-to-1 input filename(s)

logging: boolean

enable file logging

logmode: 'a' or 'w'

stwcs.wcsutil.headerlet.**create_headerlet**(**args*, ***kw*)

Create a headerlet from a WCS in a science file If both wcskey and wcsname are given they should match, if not raise an Exception

Parameters

filename: string or HDUList

Either a filename or PyFITS HDUList object for the input science file An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

sciext: string or python list (default: 'SCI')

Extension in which the science data with the linear WCS is. The headerlet will be created from these extensions. If string - a valid EXTNAME is expected If int - specifies an extension with a valid WCS, such as 0 for a simple FITS file If list - a list of FITS extension numbers or strings representing extension tuples, e.g. ('SCI, 1') is expected.

hdrname: string

value of HDRNAME keyword Takes the value from the HDRNAME<wcskey> keyword, if not available from WCSNAME<wcskey> It stops if neither is found in the science file and a value is not provided

destim: string or None

name of file this headerlet can be applied to if None, use ROOTNAME keyword

wcskey: char (A...Z) or " " or "PRIMARY" or None

a char representing an alternate WCS to be used for the headerlet if " ", use the primary (default) if None use wcsname

wcsname: string or None

if wcskey is None use wcsname specified here to choose an alternate WCS for the headerlet

sipname: string or None (default)

Name of unique file where the polynomial distortion coefficients were read from. If None, the behavior is: The code looks for a keyword 'SIPNAME' in the science header If not found, for HST it defaults to 'IDCTAB' If there is no SIP model the value is 'NOMODEL' If there is a SIP model but no SIPNAME, it is set to 'UNKNOWN'

npolfile: string or None (default)

Name of a unique file where the non-polynomial distortion was stored. If None: The code looks for 'NPOLFILE' in science header. If 'NPOLFILE' was not found and there is no npol model, it is set to 'NOMODEL' If npol model exists, it is set to 'UNKNOWN'

d2imfile: string

Name of a unique file where the detector to image correction was If None: The code looks for 'D2IMFILE' in the science header. If 'D2IMFILE' is not found and there is no d2im correction, it is set to 'NOMODEL' If d2im correction exists, but 'D2IMFILE' is missing from science header, it is set to 'UNKNOWN'

author: string

Name of user who created the headerlet, added as 'AUTHOR' keyword to headerlet PRIMARY header

descrip: string

Short description of the solution provided by the headerlet This description will be added as the single 'DESCRIP' keyword to the headerlet PRIMARY header

history: filename, string or list of strings

Long (possibly multi-line) description of the solution provided by the headerlet. These comments will be added as 'HISTORY' cards to the headerlet PRIMARY header If filename is specified, it will format and attach all text from that file as the history.

nmatch: int (optional)

Number of sources used in the new solution fit

catalog: string (optional)

Astrometric catalog used for headerlet solution

logging: boolean

enable file logging

logmode: 'w' or 'a'

log file open mode

Returns

Headerlet object

stwcs.wcsutil.headerlet.**delete_headerlet**(**args*, ***kw*)

Deletes HeaderletHDU(s) with same HDRNAME from science files

Parameters

filename: string, HDUList or list of strings

Filename can be specified as a single filename or HDUList, or a list of filenames Each input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

hdrname: string or None

HeaderletHDU primary header keyword HDRNAME

hdrext: int, tuple or None

HeaderletHDU FITS extension number tuple has the form ('HDRLET', 1)

distname: string or None

distortion model as specified in the DISTNAME keyword

logging: boolean

enable file logging

logmode: 'a' or 'w'

Notes

One of hdrname, hdrext or distname should be given. If hdrname is given - delete a HeaderletHDU with a name HDRNAME from fobj. If hdrext is given - delete HeaderletHDU in extension. If distname is given - deletes all HeaderletHDUs with a specific distortion model from fobj. Updates wcscorr

`stwcs.wcsutil.headerlet.`**`extract_headerlet`**(**args*, ***kw*)

Finds a headerlet extension in a science file and writes it out as a headerlet FITS file.

If both hdrname and extnum are given they should match, if not raise an Exception

Parameters

filename: string or HDUList or Python list

This specifies the name(s) of science file(s) from which headerlets will be extracted.

String input formats supported include use of wild-cards, IRAF-style '@'-files (given as '@<filename>') and comma-separated list of names. An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename. If a list of filenames has been specified, it will extract a headerlet from the same extnum from all filenames.

output: string

Filename or just rootname of output headerlet FITS file If string does not contain '.fits', it will create a filename with '_hlet.fits' suffix

extnum: int

Extension number which contains the headerlet to be written out

hdrname: string

Unique name for headerlet, stored as the HDRNAME keyword It stops if a value is not provided and no extnum has been specified

clobber: bool

If output file already exists, this parameter specifies whether or not to overwrite that file [Default: False]

logging: boolean

enable logging to a file

stwcs.wcsutil.headerlet.**find_headerlet_HDUs**(**args*, ***kw*)

Returns all HeaderletHDU extensions in a science file that matches the inputs specified by the user. If no hdrext, hdrname or distname are specified, this function will return a list of all HeaderletHDU objects.

Parameters

fobj : str, `astropy.io.fits.HDUList`

Name of FITS file or open fits object (`astropy.io.fits.HDUList` instance)

hdrext : int, tuple or None

index number(EXTVER) or extension tuple of HeaderletHDU to be returned

hdrname : string

value of HDRNAME for HeaderletHDU to be returned

distname : string

value of DISTNAME for HeaderletHDUs to be returned

strict : bool [Default: True]

Specifies whether or not at least one parameter needs to be provided If False, all extension indices returned if hdrext, hdrname and distname are all None. If True and hdrext, hdrname, and distname are all None, raise an Exception requiring one to be specified.

logging : boolean

enable logging to a file called headerlet.log

logmode : 'w' or 'a'

log file open mode

Returns

hdrlets : list

A list of all matching HeaderletHDU extension indices (could be just one)

stwcs.wcsutil.headerlet.**get_extname_extver_list**(*fobj*, *sciext*)

Create a list of (EXTNAME, EXTVER) tuples

Based on sciext keyword (see docstring for create_headerlet) walk throughh the file and convert extensions in `sciext` to valid (EXTNAME, EXTVER) tuples.

stwcs.wcsutil.headerlet.**get_header_kw_vals**(*hdr*, *kwname*, *kwval*, *default=0*)

stwcs.wcsutil.headerlet.**get_headerlet_kw_names**(*fobj*, *kw='HDRNAME'*)

Returns a list of specified keywords from all HeaderletHDU extensions in a science file.

Parameters

fobj : str, `astropy.io.fits.HDUList`

kw : str

Name of keyword to be read and reported

stwcs.wcsutil.headerlet.**headerlet_summary**(*filename*, *columns=None*, *pad=2*, *maxwidth=None*, *output=None*, *clobber=True*, *quiet=False*)

Print a summary of all HeaderletHDUs in a science file to STDOUT, and optionally to a text file The summary includes: HDRLET_ext_number HDRNAME WCSNAME DISTNAME SIPNAME NPOLFILE D2IMFILE

Parameters

filename: string or HDUList

Either a filename or PyFITS HDUList object for the input science file An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

columns: list

List of headerlet PRIMARY header keywords to report in summary By default (set to None), it will use the default set of keywords defined as the global list DEFAULT_SUMMARY_COLS

pad: int

Number of padding spaces to put between printed columns [Default: 2]

maxwidth: int

Maximum column width(not counting padding) for any column in summary By default (set to None), each column's full width will be used

output: string (optional)

Name of optional output file to record summary. This filename can contain environment variables. [Default: None]

clobber: bool

If True, will overwrite any previous output file of same name

quiet: bool

If True, will NOT report info to STDOUT

stwcs.wcsutil.headerlet.**init_logging**(*funcname=None*, *level=100*, *mode='w'*, ***kwargs*)

Initialize logging for a function

Parameters

funcname: string

Name of function which will be recorded in log

level: int, or bool, or string

int or string : Logging level bool: False - switch off logging Text logging level for the message ("DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL")

mode: 'w' or 'a'

attach to logfile ('a' or start a new logfile ('w')

stwcs.wcsutil.headerlet.**is_par_blank**(*par*)

stwcs.wcsutil.headerlet.**parse_filename**(*fname*, *mode='readonly'*)

Interprets the input as either a filename of a file that needs to be opened or a PyFITS object.

Parameters

fname : str, `astropy.io.fits.HDUList`

Input pointing to a file or `astropy.io.fits.HDUList` object. An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

mode : string

Specifies what mode to use when opening the file, if it needs to open the file at all [Default: 'readonly']

Returns

fobj : `astropy.io.fits.HDUList`

FITS file handle for input

fname : str

Name of input file

close_fobj : bool

Flag specifying whether or not fobj needs to be closed since it was opened by this function. This allows a program to know whether they need to worry about closing the FITS object as opposed to letting the higher level interface close the object.

`stwcs.wcsutil.headerlet.`**`print_summary`**(*summary_cols*, *summary_dict*, *pad=2*, *maxwidth=None*, *idcol=None*, *output=None*, *clobber=True*, *quiet=False*)

Print out summary dictionary to STDOUT, and possibly an output file

`stwcs.wcsutil.headerlet.`**`restore_all_with_distname`**(**args*, ***kw*)

Restores all HeaderletHDUs with a given distortion model as alternate WCSs and a primary

Parameters

filename: string or HDUList

Either a filename or PyFITS HDUList object for the input science file
An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

distname: string

distortion model as represented by a DISTNAME keyword

primary: int or string or None

HeaderletHDU to be restored as primary if int - a fits extension if string - HDRNAME if None - use first HeaderletHDU

archive: boolean (default True)

flag indicating if HeaderletHDUs should be created from the primary and alternate WCSs in fname before restoring all matching headerlet extensions

logging: boolean

enable file logging

logmode: 'a' or 'w'

`stwcs.wcsutil.headerlet.`**`restore_from_headerlet`**(**args*, ***kw*)

Restores a headerlet as a primary WCS

Parameters

filename: string or HDUList

Either a filename or PyFITS HDUList object for the input science file
An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

hdrname: string

HDRNAME keyword of HeaderletHDU

hdrext: int or tuple

Headerlet extension number of tuple ('HDRLET',2)

archive: boolean (default: True)

When the distortion model in the headerlet is the same as the distortion model of the science file, this flag indicates if the primary WCS should be saved as an alternate nd a headerlet extension. When the distortion models do not match this flag indicates if the current primary and alternate WCSs should be archived as headerlet extensions and alternate WCS.

force: boolean (default:False)

When the distortion models of the headerlet and the primary do not match, and archive is False, this flag forces an update of the primary.

logging: boolean

enable file logging

logmode: 'a' or 'w'

stwcs.wcsutil.headerlet.**update_ref_files**(*source*, *dest*)
Update the reference files name in the primary header of 'dest' using values from 'source'

Parameters
source : astropy.io.fits.Header

dest : astropy.io.fits.Header

stwcs.wcsutil.headerlet.**update_versions**(*sourcehdr*, *desthdr*)
Update keywords which store version numbers

stwcs.wcsutil.headerlet.**verify_hdrname_is_unique**(*fobj*, *hdrname*)
Verifies that no other HeaderletHDU extension has the specified hdrname.

Parameters
fobj : str, astropy.io.fits.HDUList

Name of FITS file or open fits file object

hdrname : str

value of HDRNAME for HeaderletHDU to be compared as unique

Returns
unique: bool

If True, no other HeaderletHDU has the specified HDRNAME value

stwcs.wcsutil.headerlet.**with_logging**(*func*)

stwcs.wcsutil.headerlet.**write_headerlet**(**args*, ***kw*)
Save a WCS as a headerlet FITS file.

This function will create a headerlet, write out the headerlet to a separate headerlet file, then, optionally, attach it as an extension to the science image (if it has not already been archived)

Either wcsname or wcskey must be provided; if both are given, they must match a valid WCS.

Updates wcscorr if necessary.

Parameters

filename: string or HDUList or Python list

This specifies the name(s) of science file(s) from which headerlets will be created and written out. String input formats supported include use of wild-cards, IRAF-style '@'-files (given as '@<filename>') and comma-separated list of names. An input filename (str) will be expanded as necessary to interpret any environmental variables included in the filename.

hdrname: string

Unique name for this headerlet, stored as HDRNAME keyword

output: string or None

Filename or just rootname of output headerlet FITS file If string does not contain '.fits', it will create a filename starting with the science filename and ending with '_hlet.fits'. If None, a default filename based on the input filename will be generated for the headerlet FITS filename

sciext: string

name (EXTNAME) of extension that contains WCS to be saved

wcsname: string

name of WCS to be archived, if " ": stop

wcskey: one of A...Z or " " or "PRIMARY"

if " " or "PRIMARY" - archive the primary WCS

destim: string

DESTIM keyword if NOne, use ROOTNAME or science file name

sipname: string or None (default)

Name of unique file where the polynomial distortion coefficients were read from. If None, the behavior is: The code looks for a keyword 'SIPNAME' in the science header If not found, for HST it defaults to 'IDCTAB' If there is no SIP model the value is 'NOMODEL' If there is a SIP model but no SIPNAME, it is set to 'UNKNOWN'

npolfile: string or None (default)

Name of a unique file where the non-polynomial distortion was stored. If None: The code looks for 'NPOLFILE' in science header. If 'NPOLFILE' was not found and there is no npol model, it is set to 'NOMODEL' If npol model exists, it is set to 'UNKNOWN'

d2imfile: string

Name of a unique file where the detector to image correction was stored. If None: The code looks for 'D2IMFILE' in the science header. If 'D2IMFILE' is not found and there is no d2im correction, it is set to 'NOMODEL' If d2im correction exists, but 'D2IMFILE' is missing from science header, it is set to 'UNKNOWN'

author: string

Name of user who created the headerlet, added as 'AUTHOR' keyword to headerlet PRIMARY header

descrip: string

Short description of the solution provided by the headerlet This description will be added as the single 'DESCRIP' keyword to the headerlet PRIMARY header

history: filename, string or list of strings

Long (possibly multi-line) description of the solution provided by the headerlet. These comments will be added as 'HISTORY' cards to the headerlet PRIMARY header If filename is specified, it will format and attach all text from that file as the history.

attach: bool

Specify whether or not to attach this headerlet as a new extension It will verify that no other headerlet extension has been created with the same 'hdrname' value.

clobber: bool

If output file already exists, this parameter specifies whether or not to overwrite that file [Default: False]

logging: boolean

enable file logging

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

[R1] Jay Anderson, “Variation of the Distortion Solution of the WFC”, ACS ISR 2007-08.

[R4] David Shupe, et al, “The SIP Convention of representing Distortion in FITS Image headers”, Astronomical Data Analysis Software And Systems, ASP Conference Series, Vol. 347, 2005

S