STWCS Documentation

Release 1.7.2.dev8+gd389a1d

Nadezhda Dencheva, Warren Hack

Nov 29, 2021

Contents

1 wcsutil **3**
- 1.1 HSTWCS API . . . 3
- 1.2 HSTWCS Examples . . . 8
- 1.3 User Interface: altwcs . . . 9

2 updatewcs **13**
- 2.1 UPDATEWCS - User Interface . . . 14
- 2.2 WCS Corrections . . . 15
- 2.3 UPDATEWCS.UTILS functions . . . 17
- 2.4 Astrometry Database Interface . . . 17

3 Headerlet **21**
- 3.1 Headerlet File Structure . . . 21
- 3.2 User-Interface: headerlet . . . 22

4 Technical reports **35**
- 4.1 TSR 2012-01: Distortion Correction in HST FITS Files . . . 35
- 4.2 TSR 2012-03: NPOL Reference File . . . 53
- 4.3 TSR 2012-02: Definition of a Headerlet and Its Role in Updating WCS Information . . . 61

5 Indices and tables **79**

Bibliography **81**

Python Module Index **83**

Index **85**

This package provides support for WCS based distortion models and coordinate transformation developed to support HST imaging observations. It consists of two subpackages - updatewcs and wcsutil. updatewcs performs corrections to the basic WCS and inserts other distortion infomation in the science files as header keywords or file extensions. wcsutil provides an `wcsutil.STWCS` object which extends astropy's `astropy.wcs.WCS` provides HST instrument specific information as well as methods for coordinate transformation. wcsutil also provides functions for manipulating alternate WCS descriptions in the headers. STWCS is based on astropy.wcs.

Contents:

CHAPTER 1

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=' '*)

all_world2pix(**args*, ***kwargs*)

all_world2pix(*arg, tolerance=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

- **tolerance** (*float, optional (Default = 1.0e-4)*) – Absolute tolerance required 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 absolute tolerance 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, tolerance=1.0e-10, quiet=False)
NoConvergence: 'HSTWCS.all_world2pix' failed to converge to requested␣
↪accuracy after 3 iterations.
```

```
>>>
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, tolerance=1.0e-4,␣
↪adaptive=False, detect_divergence=True, quiet=False)
>>> 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, tolerance=1.0e-4,
      adaptive=False, detect_divergence=False, quiet=False)
>>> 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]
```

(continues on next page)

(continued from previous page)

```
 [  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.
```

naxis1

naxis2

pc2cd()

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

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.

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

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

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*)

Deprecated since version 1.6.0: The archiveWCS function is deprecated and may be removed in a future version. Use archive_wcs instead.

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** (*bool*) – if True - overwrites a WCS with the same key

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']
```

See also:

wcsutil.restoreWCS() Copy an alternate WCS to the primary WCS

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*, *include_primary=True*)
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 2

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*, *use_db=True*, *all_wcs=False*, *remove_duplicates=True*)

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. Duplicate HeaderletHDU extensions (each containing a separate WCS) can also be deleted from the file as well. These duplicates generally are unintended, but if left in place, they can cause an Exception to be thrown when the user works with these extensions later.

Examples

```
>>> from stwcs import updatewcs
>>> updatewcs.updatewcs(filename)
```

Dependencies

stsci.tools astropy.io.fits astropy.wcs requests lxml

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** (*bool*) – If True, vecocity aberration correction will be applied
- **tddcorr** (*bool*) – If True, time dependent distortion correction will be applied
- **npolcorr** (*bool*) – If True, a Lookup table distortion will be applied
- **d2imcorr** (*bool*) – If True, detector to image correction will be applied
- **checkfiles** (*bool*) – 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.
- **use_db** (*bool*) – If True, attempt to add astrometric solutions from the MAST astrometry database. Default value is True.
- **all_wcs** (*bool*) – This parameter only gets used if use_db=True to control what WCS solutions from the Astrometry database gets appended to the file. If True, all solutions get appended. If False, only solutions based on the IDCTAB from the file's PRIMARY header will be appended.
- **remove_duplicates** (*bool, optional*) – This parameter only gets used if use_db=True to remove any duplicate headerlet extensions. These extensions contain WCS solutions that are identical to the WCS found in other extensions of the image.

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[1] 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

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.

[1] Jay Anderson, "Variation of the Distortion Solution of the WFC", ACS ISR 2007-08.

2.2.3 Simple Imaging Polynomial Coefficients

class stwcs.updatewcs.corrections.**CompSIP**

Compute Simple Imaging Polynomial (SIP) coefficients as defined in[2] 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

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] 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

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

Return type 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

Return type 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

Return type d2imname, d2imfile

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

2.4 Astrometry Database Interface

This module provides an interface for accessing a database which contains updated WCS solutions for many images in the HST archive. The exact set of images with new solutions in this database will change over time, with the initial release containing updates only for ACS and WFC3 direct images. This database will be designed to provide multiple WCS solutions for each exposure depending on what astrometric frame was used to define the new WCS solution; for example, an image may be matched to Pan-STARRS sources in one solution and to GAIA sources in another solution.

Each new WCS solution will be appended to the exposure by this module as a headerlet extension, taking advantage of the headerlet definition provided by the STWCS package, found in *Headerlet* documentation. The headerlet library allows a user to manage all appended solutions and determine which one gets used by the exposure at anytime, since only 1 WCS can be used at a time.

2.4.1 Usage

This interface defines an object that provides the services for updating an exposure with new WCS solutions. Only 2 calls are typically necessary to update any exposure or set of exposures.

0. Import the package

```
>>> from stwcs.updatewcs import astrometry_utils as stwcsau
```

1. Establish a working connection with the database

```
>>> db = stwcsau.AstrometryDB()
```

2. For each exposure, query the database and update with any new WCS solutions

```
>>> db.updateObs(filename)
```

The environment variables used by this module can be defined by the user to change what database is being accessed, whether to raise exceptions instead of warnings when problems arise in connecting to the database or otherwise updating an exposure, or to completely turn on/off use of this module. These allow external code, such as updatewcs (see *updatewcs*), to include this code and control it as necessary.

2.4.2 API

Interface for Astrometry database service.

This module contains interface functions for the AstrometryDB Restful service based on interfaces/code provided by B. McLean 11-Oct-2017.

The code checks for the existence of environmental variables to help control the operation of this interface; namely,

RAISE_PIPELINE_ERRORS - boolean to specify whether to raise exceptions during processing or simply log errors and quit gracefully. If not set, default behavior will be to log errors and quit gracefully.

ASTROMETRY_SERVICE_URL - URL pointing to user-specified web service that will provide updated astrometry solutions for observations being processed. This will replace the built-in URL included in the base class. This value will also be replaced by any URL provided by the user as an input parameter `url`.

ASTROMETRY_STEP_CONTROL - String specifying whether or not to perform the astrometry update processing at all. Valid Values: "ON", "On", "on", "OFF", "Off", "off" If not set, default value is "ON".

GSSS_WEBSERVICES_URL - URL point to user-specified web service which provides information on the guide stars used for taking HST observations. This value will replace the default URL included in this module as the `gsss_url` variable.

class stwcs.updatewcs.astrometry_utils.**AstrometryDB**(*url=None*, *raise_errors=None*, *perform_step=True*, *write_log=False*)

Bases: `object`

Base class for astrometry database interface.

apply_new_apriori(*obsname*)

Compute and apply a new a priori WCS based on offsets from astrometry database.

Parameters **obsname** (*str*) – Full filename or `astropy.io.fits.HDUList` object for the observation to be corrected

Returns **wcsname** – Value of WCSNAME keyword for this new WCS

Return type str

findObservation(*observationID*)
Find whether there are any entries in the AstrometryDB for the observation with `observationID`.

Parameters **observationID** (*str*) – base rootname for observation to be updated (eg., `iab001a1q`)

Returns **entry** – Database entry for this observation, if found. It will return None if there was an error in accessing the database and `self.raise_errors` was not set to True.

Return type obj

getObservation(*observationID*)
Get solutions for observation from AstrometryDB.

Parameters **observationID** (*str*) – base rootname for observation to be updated (eg., `iab001a1q`)

Returns

- **headerlets** (*dict*) – Dictionary containing all solutions found for exposure in the form of headerlets labelled by the name given to the solution in the database.
- **best_solution_id** (*str*) – WCSNAME of the WCS solution flagged as 'best' in the astrometry database for the observation. The 'best' solution should be the one that aligns the observation as close to the GAIA frame as possible.

isAvailable()
Test availability of astrometryDB web-service.

updateObs(*obsname*, *all_wcs=False*, *remove_duplicates=True*)
Update observation with any available solutions.

Parameters

- **obsname** (*str*) – Filename for observation to be updated
- **all_wcs** (*bool*) – If True, all solutions from the Astrometry database are appended to the input file as separate FITS extensions. If False, only those solutions based on the same IDCTAB will be appended.
- **remove_duplicates** (*bool*) – If True, any headerlet extensions with the same HDRNAME are found, the copies will be deleted until only the first version added remains.

stwcs.updatewcs.astrometry_utils.**apply_astrometric_updates**(*obsnames*, ***pars*)
Apply new astrometric solutions to observation.

Functional stand-alone interface for applying new astrometric solutions found in the astrometry dB to the given observation(s).

Parameters

- **obsnames** (*str, list*) – Filename or list of filenames of observation(s) to be updated
- **url** (*str, optional*) – URL of astrometry database web-interface to use. If None (default), it will use built-in URL for STScI web-interface
- **raise_errors** (*bool, optional*) – Specify whether or not to raise Exceptions and stop processing when an error in either accessing the database, retrieving a solution from the database or applying the new solution to the observation. If None, it will look to see whether the environmental variable `RAISE_PIPELINE_ERRORS` was set, otherwise, it will default to 'False'.

2.4.3 Version

stwcs.**__version__ = '1.7.2.dev8+gd389a1d'**
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

CHAPTER 3

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.

NOTE :: This module defines a FileHandler for the logging in the current working directory for the user when this module first gets imported. If that directory is removed later by the user, it will cause an Exception when performing headerlet operations later.

The file handler can be identified using:

```
rl = logging.getLogger('stwcs.wcsutil.headerlet')
rl.handlers
del rl.handlers[-1]  # if FileHandler was the last one, remove it
```

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

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

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

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** (*bool*) – 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** (*bool*) – 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** (*bool (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** (*bool (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*)
Creates an `HDUList` instance from a file-like object.

The actual implementation of `fitsopen()`, and generally shouldn't be used directly. Use `open()` instead (and see its documentation for details of the parameters accepted by this method).

classmethod fromstring(*data*, ***kwargs*)
Creates an `HDUList` instance from a string or other in-memory data buffer containing an entire FITS file. Similar to `HDUList.fromfile()`, but does not accept the mode or memmap arguments, as they are only relevant to reading from a file on disk.

This is useful for interfacing with other libraries such as CFITSIO, and may also be useful for streaming applications.

Parameters

- **data** (*str, buffer-like, etc.*) – A string or other memory buffer containing an entire FITS file. Buffer-like objects include `bytes`, `bytearray`, `memoryview`, and `ndarray`. It should be noted that if that memory is read-only (such as a Python string) the returned `HDUList`'s data portions will also be read-only.
- **kwargs** (*dict*) – Optional keyword arguments. See `astropy.io.fits.open()` for details.

Returns **hdul** – An `HDUList` object representing the in-memory FITS file.

Return type HDUList

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** – Dictionary of values for summary

Return type dict

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** (*bool*) – 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*, *ver=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** – A *HeaderletHDU* object for the given *Headerlet* that can be attached as an extension to an existing HDUList.

Return type *HeaderletHDU*

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**(*filename*, *hdrlet*, *attach=True*, *wcskey=None*, *wcsname=None*, *logging=False*, *logmode='w'*)

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** (*bool*) – 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** (*bool*) – enable file logging
- **logmode** (*'a' or 'w'*) –

stwcs.wcsutil.headerlet.**apply_headerlet_as_primary**(*filename*, *hdrlet*, *attach=True*, *archive=True*, *force=False*, *logging=False*, *logmode='a'*)

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** (*bool*) – True (default): append headerlet to FITS file as a new extension.
- **archive** (*bool*) – True (default): before updating, create a headerlet with the WCS old solution.
- **force** (*bool*) – If True, this will cause the headerlet to replace the current PRIMARY WCS even if it has a different distortion model. [Default: False]
- **logging** (*bool*) – enable file logging
- **logmode** (*'w' or 'a'*) – log file open mode

stwcs.wcsutil.headerlet.**archive_as_headerlet**(*filename*, *hdrname*, *sciext='SCI'*, *wcsname=None*, *wcskey=None*, *destim=None*, *sipname=None*, *npolfile=None*, *d2imfile=None*, *author=None*, *descrip=None*, *history=None*, *nmatch=None*, *catalog=None*, *logging=False*, *logmode='w'*)

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 multiline) 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** (*bool*) – enable file folling
- **logmode** (*'w' or 'a'*) – log file open mode

stwcs.wcsutil.headerlet.**attach_headerlet**(*filename*, *hdrlet*, *logging=False*, *logmode='a'*)

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** (*bool*) – enable file logging
- **logmode** (*'a' or 'w'*) –

stwcs.wcsutil.headerlet.**create_headerlet**(*filename*, *sciext='SCI'*, *hdrname=None*, *destim=None*, *wcskey=' '*, *wcsname=None*, *sipname=None*, *npolfile=None*, *d2imfile=None*, *author=None*, *descrip=None*, *history=None*, *nmatch=None*, *catalog=None*, *logging=False*, *logmode='w'*)

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 multiline) 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** (*bool*) – enable file logging
- **logmode** (*'w' or 'a'*) – log file open mode

Returns

Return type Headerlet object

stwcs.wcsutil.headerlet.**delete_headerlet**(*filename*, *hdrname=None*, *hdrext=None*, *distname=None*, *keep_first=False*, *logging=False*, *logmode='w'*)

Deletes all HeaderletHDUs with same HDRNAME from science files

Notes

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

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
- **keep_first** (*bool, optional*) – If True, the first matching HeaderletHDU found will be NOT deleted.
- **logging** (*bool*) – enable file logging
- **logmode** (*'a' or 'w'*) –

stwcs.wcsutil.headerlet.**extract_headerlet**(*filename*, *output*, *extnum=None*, *hdrname=None*, *clobber=False*, *logging=True*)

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** (*bool*) – enable logging to a file

stwcs.wcsutil.headerlet.**find_headerlet_HDUs**(*fobj*, *hdrext=None*, *hdrname=None*, *distname=None*, *strict=True*, *logging=False*, *logmode='w'*)

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** (*bool*) – enable logging to a file called headerlet.log

- **logmode** (*'w' or 'a'*) – log file open mode

Returns **hdrlets** – A list of all matching HeaderletHDU extension indices (could be just one)

Return type list

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**(*filename*, *distname*, *primary*, *archive=True*, *sciext='SCI'*, *logging=False*, *logmode='w'*)

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** (*bool (default True)*) – flag indicating if HeaderletHDUs should be created from the primary and alternate WCSs in fname before restoring all matching headerlet extensions
- **logging** (*bool*) – enable file logging
- **logmode** (*'a' or 'w'*) –

stwcs.wcsutil.headerlet.**restore_from_headerlet**(*filename*, *hdrname=None*, *hdrext=None*, *archive=True*, *force=False*, *logging=False*, *logmode='w'*)

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** (*bool (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** (*bool (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** (*bool*) – 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** – If True, no other HeaderletHDU has the specified HDRNAME value

Return type bool

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

stwcs.wcsutil.headerlet.**write_headerlet**(*filename*, *hdrname*, *output=None*, *sciext='SCI'*, *wcsname=None*, *wcskey=None*, *destim=None*, *sipname=None*, *npolfile=None*, *d2imfile=None*, *author=None*, *descrip=None*, *history=None*, *nmatch=None*, *catalog=None*, *attach=True*, *clobber=False*, *logging=False*)
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** (*bool*) – enable file logging

CHAPTER 4

Technical reports

4.1 TSR 2012-01: Distortion Correction in HST FITS Files

Abstract: Authors: Warren Hack, Nadezhda Dencheva, Andy Fruchter, Perry Greenfield Date: 31 Oct 2012

A convention for storing distortion information in HST images was developed and implemented in two software packages - PyWCS and STWCS. These changes allow the development of a WCS based version of Multidrizzle, released now as AstroDrizzle, and image alignment software, released as tweakreg. The distribution of WCS solutions is discussed.

Contents:

4.1.1 Introduction

Calibration of the HST Advanced Camera for Surveys (HST/ACS) distortion requires the use of several components to the distortion correction; namely, polynomial coefficients, a correction for velocity aberration, a time-dependent skew, a lookup table for non-polynomial terms, and a detector defect correction. Each of these terms has been derived as part of the calibration effort to address separate aspects of the distortion that affects ACS observations. Ideally, each would be applied independently in the same manner used for deriving the original calibration reference information, with the time-dependent skew being folded into the other terms. However, the software for applying the distortion models does not support this option. In fact, there is no clear accepted standard for specifying distortion corrections in FITS headers. Instead, there are several separate proposals for specifying aspects of the distortion, but none by themselves allows us to fully specify the distortion already calibrated for ACS, let alone in a modular, efficient manner.

This paper describes a composite implementation of a select set of proposed standards which supports all aspects of the distortion models for HST instruments without breaking any of the conventions/standards. The rules for merging the proposed standards allow software to be defined to apply each aspect of this proposal as a separate option while defining the requirements necessary to allow them to work together when specified in the header. As a result, the separate components essentially become tools where only those conventions appropriate to the observation can be used as needed.

4.1.2 Problems Introduced by the HST/ACS Distortion

All calibrations for HST observations get recorded and applied through the use of reference files, separate files which describe some calibration. The geometric distortion typically applied to HST images gets recorded as a polynomial component in one reference file, and a pixel-by-pixel correction to the polynomial solution in a separate reference file. This method allows the distortion to be corrected to an accuracy of better than 0.1 pixels. However, this method requires the user to obtain the reference files themselves anytime they want to reprocess the data. The size of these reference files (up to 200Mb) makes this an expensive requirement for the end user. The alternative would be to include the necessary specification of the distortion model in the header of the science image itself, as long as it can be done in a manner that does not dramatically increase the size of the FITS file. For reference, a typical calibrated ACS/WFC image requires a 168Mb file. Thus, we needed an alternative to separate reference files which can be implemented in a very efficient manner within the image's FITS headers.

The calibrations also represent separate aspects of the detector and the distortion, aspects which logically should remain as separate descriptions in the header. The pixels for each CCD do not have the same size across the entire chip. The ACS/WFC CCDs manufacturing process resulted in the pixels having different sizes every 68.3 columns, and can be represented most efficiently and accurately by a 1-D correction that gets applied to every row in the chip. This detector level characterization affects all images readout from the CCD regardless of any additional distortion applied to the field of view. Logically, this effect should be kept as a separate component of the distortion model that gets applied prior to correcting for any other distortion. This represents an example of an effect that is best applied sequentially to the image data.

Additional distortions come as a result of the effect of the optics on the field of view. These are generally described by low-order polynomials for most instruments, although for ACS, an additional non-polynomial correction needed to be taken into account as well. Fortunately, the non-polynomial correction can sub-sampled enough to make it practical to include in the image headers directly, a correction of up to a quarter of a pixel in some areas of the detector. Both components need to be applied to the data in order to align images well enough for subsequent data analysis or cosmic-ray rejection.

These corrections could be combined into a single look-up table, yet it would come at the cost of additional errors which may not allow the remaining data analysis or cosmic-ray rejection to actually succeed. We also have some instruments where there is only a polynomial component, requiring the development of support for a polynomial correction and a look-up table anyway.

These requirements on the application of the calibrations to HST data leave us with no alternative within current FITS standards. As a result, we developed this set of rules which allow us to take advantage of the most appropriate conventions for each separate component of the distortion model and combine them in an efficient manner which eliminates the need for external reference data.

4.1.3 SIP Convention

Current implementations of distortion models in FITS headers have been limited to simply describing polynomial models. The prime example of this would be the implementation of SIP in WCSTOOLS and recognition of SIP keywords by DS9 as used for Spitzer data [SIPConvention]. The new keywords defined by the SIP standard and used by PyWCS are:

```
A_ORDER =    n  / polynomial order, axis 1, detector to sky
A_i_j           / High order coefficients for X axis
B_ORDER =    m  /  polynomial order, axis 2, detector to sky
B_i_j           / High order coefficients for axis 2
```

These SIP keywords get used in conjunction with the linear WCS keywords defined with these values:

```
CTYPE1  = 'RA---TAN-SIP'
CTYPE2  = 'DEC--TAN-SIP'
CDi_j           / Linear terms of distortion plus scale and orientation
```

The SIP convention retains the use of the current definition of the CD matrix where the linear terms of the distortion model are folded in with the orientation and scale at the reference point for each chip to provide the best linear approximation to the distortion available. The SIP convention gets applied to the input pixel positions by applying the higher-order coefficients A_i_j, B_i_j, then by applying the CD matrix and adding the CRVAL position to get the final world coordinates.

This convention was created from the original form of the FITS Distortion Paper standards, but the FITS Distortion Paper proposal has since changed to use a different set of keywords and conventions.

A sample ACS/WFC SCI header can be found in *Appendix 1: Headerlet API* to illustrate how these keywords actually get populated for an image. The current implementation does not take advantage of the A_DMAX, B_DMAX, SIPREFi or SIPSCLi keywords, so these keywords are not written out to the SCI header.

Velocity Aberration Correction

This correction simply serves as a correction to the overall linear scale of the field of view due to velocity aberration observed due to the motion of HST in orbit. The typical plate scale for HST cameras results in a measurable velocity aberration with variations from the center of the field of view to the edge on the order of 0.1 pixels. More details about this correction can be found in Appendix A.3 of the DrizzlePac Handbook.

This scale factor gets computed by the HST ground systems for start of each exposure and recorded as the VAFACTOR keyword in each image's science extension header. This term, though, does not get included in the default CD matrix computed by the ground systems. As a result, it needs to be accounted for when reading in the distortion model polynomial coefficients from the IDCTAB reference table. The VAFACTOR scaling factor gets folded into the computation of new values for the CD matrix for this specific exposure without requiring any further use of the VAFACTOR keyword when applying this distortion model to the science image. It also gets used to correct the reference position of each chip on the sky, each chip's CRVAL value, to account for this aberration.

Time-Dependent Distortion

Calibration of HST/ACS imaging data required the addition of a time dependent skew in addition to the other distortion terms. This skew represented a linear correction to the polynomial model and its residuals. This correction gets applied to the polynomial coefficients and the residuals from the polynomial model when they are evaluated for each image. As a result, the SIP keywords as written out to each HST/ACS image header reflects this time-dependent correction without the need for any further evaluation of this skew.

4.1.4 FITS Distortion Proposal

The current FITS Distortion Paper conventions [DistortionPaper] provide a mechanism for specifying either a lookup table or polynomial model for the distortion of each axis. The standard states in Section 2.1:

> Note that the prior distortion functions, $\delta_p(p)$, operate on pixel coordinates (i.e. p rather than $p - r$), and that the independent variables of the distortion functions are the *uncorrected* pixel or intermediate pixel coordinates. That is, for example, we do not allow the possibility of
>
> $$q_3' = q_3 + \delta_{q_3}(q_1', q_2') \tag{4.1}$$

The keywords used for describing these corrections use the syntax given in Table 2 of the FITS Distortion Paper. For our purposes, the keywords of interest are those related to lookup tables; namely,

```
CPDISja         string    2.4.1 distortion code new Prior distortion function type.
DPja            record    2.4.2 distortion parameter new Parameter for a prior␣
↪distortion
                                function, for use in an image header
```

This syntax only provides the option to specify one correction at a time for each axis of the image. This precludes being able to use this convention to specify both a lookup table and a polynomial model at the same time for the same axis. It does not state what should be done if the polynomial has been specified using a different convention, for example, the SIP convention. Thus, SIP and FITS Distortion Paper should not be seen as mutually exclusive. In fact, they may work together rather naturally since the SIP and FITS Distortion Paper conventions both assume the corrections will work on the input pixel and add to the output frame.

The sample header in *Appendix 1: Headerlet API* shows how these keywords get populated for an actual reference file; specifically, an NPOLFILE as described in the next section.

4.1.5 Non-polynomial Residual Correction

ACS and WFPC2 images used the DGEOFILE reference file to specify the residual correction in X and Y for each and every pixel in each chip of the observation. These DGEOFILE reference fiels required up to 168Mb each to cover all chips of each camera for ACS/WFC images. Distortion residuals have been always been calibrated for ACS by looking at the average correction that needs to be applied over each 64x64 pixel section of each chip after applying the polynomial coefficients correction. This would normally result in a 64 x 32 array of residuals for each 4096 x 2048 chip. These arrays get expanded by one value in each dimension to support interpolation all the way to the edge of each chip resulting in 65 x 33 arrays of distortion correction data.

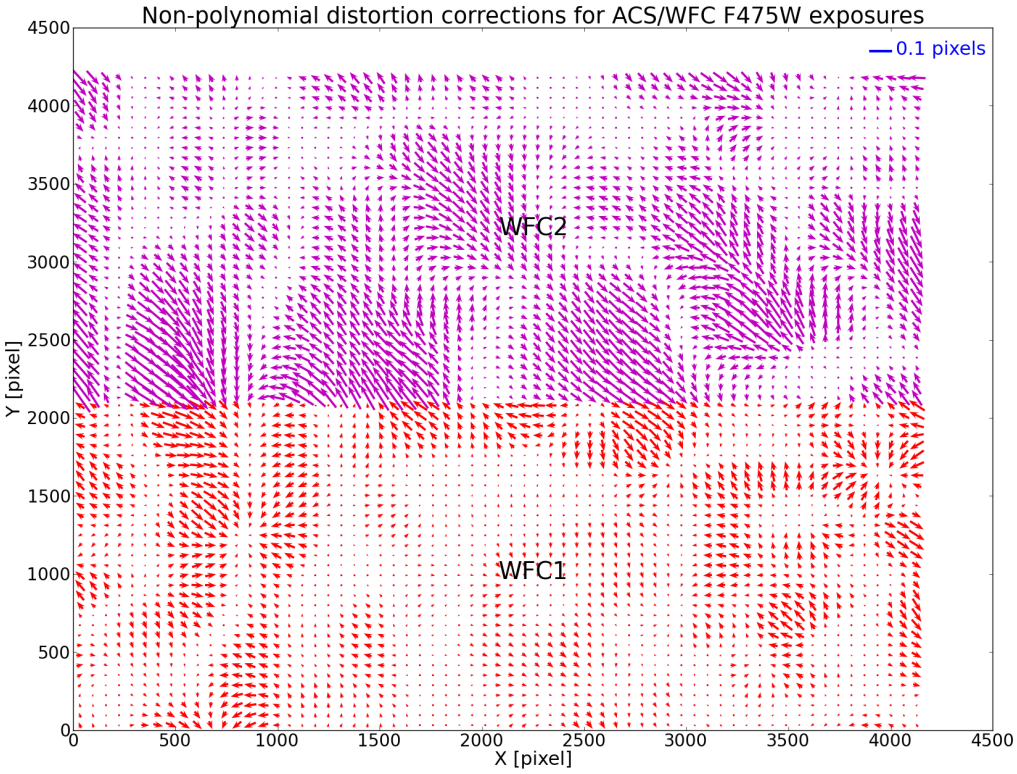


Fig. 1: This figure illustrates the corrections included in the ACS/WFC F475W non-polynomial distortion correction included in the new NPOLFILE reference file. Each vector represents the correction for a 64x64 pixel section of each chip.

These look-up tables follow the conventions in the WCS FITS Distortion Paper [DistortionPaper]. Record-valued

keywords are used to map an image in the science extension to a distortion array in the WCSDVAR extension. This new type of FITS keywords has been implemented in PyFITS and is fully described in [DistortionPaper]. Specifically, DPj.EXTVER in the science extension header maps the science image to the correct WCSDVAR extension. The dimensionality of the distortion array is defined by DPj.NAXES. Keywords DPj.AXIS.j in the SCI extension header are used for mapping image array axis to distortion array axis. In the keywords above j is an integer and denotes the axis number. For example, if distortion array axis 1 corresponds to image array axis 1 of a SCI extension, then DP.1.AXIS.1 = 1. A full example of the keywords added to a SCI extension header is presented in *Appendix 1: Headerlet API*.

A complete description of the conversion of the DGEOFILE reference data into NPOLFILE reference files can be found in the report on the npolfile-tsr.

NPOLFILE reference File Format

With the goal of including all distortion reference information directly in the science image's FITS file, including the full 168Mb DGEOFILE for ACS/WFC images would more than double the size of each input image. A new reference file based on the sub-sampled calibrations, though, would be small enough to serve as the basis for a new reference file while also being a more direct use of the calibration data. This new reference file has been called **NPOLFILE** in the FITS image header, so that any original DGEOFILE reference filename can be retained in parallel for backwards compatibility with the current software. This reference file also has a unique suffix, **_npl.fits**, as another means of identifying it as a new reference file separate from the current DGEOFILE files. The header for this new reference file also remains very simple, as illustrated in *Appendix 2 - NPOLFILE Example*.

Applying these corrections starts by reading the two 65 x 33 arrays into memory with each input ACS/WFC chip WCS (one for X offsets and one for Y offsets). Bi-linear interpolation based on the input pixel position then gets used on-the-fly to extract the final offset from this reference file. Initial versions of these sub-sampled NPOLFILE reference files for ACS have been derived from the current full-size DGEOFILEs, and testing indicates residuals only on the order of 0.02 pixels or less remain when compared to the original calibration.

4.1.6 Detector To Image Correction

The last element of the distortion which remains to be described is the fixed column (or row) width correction. This needs to be applied as a correction to the input pixel position and the output of this correction is to be used as input to the polynomial and non-polynomial distortion corrections.

The adopted implementation is based on the FITS Distortion Paper lookup table convention. It is assumed that the detector to image correction is the same for all chips but it can be extended to arbitrary number of chips and extensions if necessary.

For ACS the correction is stored as an image extension with one row. Each element in the row specifies the correction in pixels for every pixel in the column (or row) in the science extension as predetermined by the calibration teams who would be responsible for creating the reference files. For ACS the correction is in the X direction and for WFPC2 - in the Y direction. The following new keywords are added to the header of each science extension of a science file:

```
'D2IMFILE' = "string - name of reference file to be used for creating the lookup table
↪"
'AXISCORR' = "integer (1 or 2) - axis to which the det2im correction is applied"
'D2IMEXT' = "string - name of reference file which was last used to create the lookup␣
↪table"
'D2IMERR' = (optional)" float - maximum value of the correction"
```

'D2IMFILE' is used by UPDATEWCS as a flag that a reference file with this correction exists and an extension should be created. UPDATEWCS records the name of the reference file used for the lookup table extension to a keyword D2IMEXT in the primary header. It also populates keyword 'AXISCORR' based on whether this is a row or column correction. The lookup table extension has an 'EXTNAME' value of 'D2IMARR'.

'AXISCORR' is used as an indication of the axis to which the correction should be applied (1 - 'X' Axis, 2- 'Y' axis). 'D2IMEXT' stores the name of the reference file used by UPDATEWCS to create a `D2IMARR` extension. If 'D2IMEXT' is present in the 'SCI' extension header and is different from the current value of D2IMFILe in the primary header, the correction array in `D2IMARR` is updated. The optional keyword 'D2IMERR' allows a user to ignore this correction without modifying other header keywords by passing a parameter to the software. The HSTWCS class accepts a parameter 'minerr' which specifies the minimum value a distortion correction must have in order to be applied. If 'minerr' is larger than 'D2IMERR' the correction is not applied.

Detector To Image Reference File

An entirely new reference file, the D2IMFILE reference table, serves as the source of this 1-D correction for each affected instrument. This reference file only contains a single array of offsets corresponding to the 1-D correction to be applied. Header keywords in the reference file specify which axis needs this correction. As a result, this new reference file remains small enough to easily be added to an input image without significant change in size. An initial **D2IMFILE** for ACS has been generated for testing with a sample header provided in *Appendix 3 - D2IMFILE Example*.

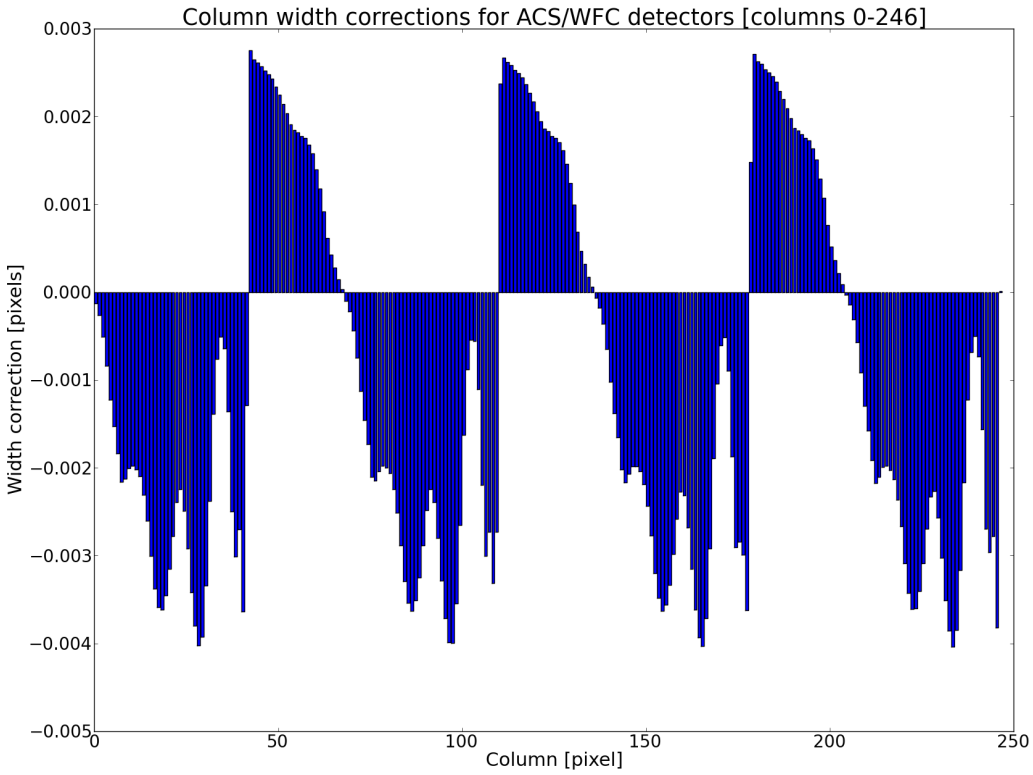


Fig. 2: This figure illustrates the corrections included in the first 246 columns of the ACS/WFC F475W D2IMFILE.

The WCS for this correction describes the extension as a 1-D image, even though it gets applied to a 2-D image. This keeps it clear that the same correction gets applied to all rows(columns) without interpolation. The header specifies which axis this correction applies to through the use of the AXISCORR keyword. The WCS keywords in the header of the `D2IMARR` extension specifies the transformation between pixel coordinates and lookup table position as if the lookup table were an image itself with 1-based positions (starting pixel is at a position of (1,1)). The value at that lookup table position then gets used to correct the original input pixel position.

4.1.7 Merging Of The Conventions

The full implementation of all these elements ends up merging the SIP, DET2IM and FITS Distortion Paper conventions to create a new version of the figure from the FITS Distortion Paper which illustrates the conversion of detector coordinates to world coordinates. This implementation works in the following way:

1. Apply detector to image correction (DET2IM) to input pixel values
2. Apply SIP coefficients to DET2IM-corrected pixel values
3. Apply lookup table correction to DET2IM-corrected pixel values
4. Add the results of the SIP and lookup table corrections
5. Apply the WCS transformation in the CD matrix to the summed results to get the intermediate world coordinates
6. Add the CRVAL keyword values to the transformed positions to get the final world coordinates

The computations to perform these steps can be described approximately using:

$$(x', y') = DET2IM(x, y) \tag{4.2}$$

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} x' - CRPIX1 \\ y' - CRPIX2 \end{pmatrix} \tag{4.3}$$

$$\begin{pmatrix} \alpha \\ \delta \end{pmatrix} = \begin{pmatrix} CRVAL1 \\ CRVAL2 \end{pmatrix} + \begin{pmatrix} CD11 & CD12 \\ CD21 & CD22 \end{pmatrix} \begin{pmatrix} u' + f(u', v') + LT_x(x', y') \\ v' + g(u', v') + LT_y(x', y') \end{pmatrix} \tag{4.4}$$

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

$$\begin{aligned} f(u', v') &= \sum_{p+q=2}^{AORDER} A_{pq} u'^p v'^q \\ g(u', v') &= \sum_{p+q=2}^{BORDER} B_{pq} u'^p v'^q \end{aligned} \tag{4.5}$$

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

These equations do not take into account the deprojection from the tangent plane to sky coordinates. The complete Detector To Sky Coordinate Transformation is based on the CTYPE keyword.

4.1.8 Updating the FITS File

Updating each science image with the distortion model using this merged convention requires integrating these new reference files directly into the FITS file. This update gets performed using the following steps:

- determining what reference files should be applied to the science image
- read in distortion coefficients from IDCTAB reference file
- [for ACS data only] compute time-dependent (TDD) skew terms from model described in IDCTAB file

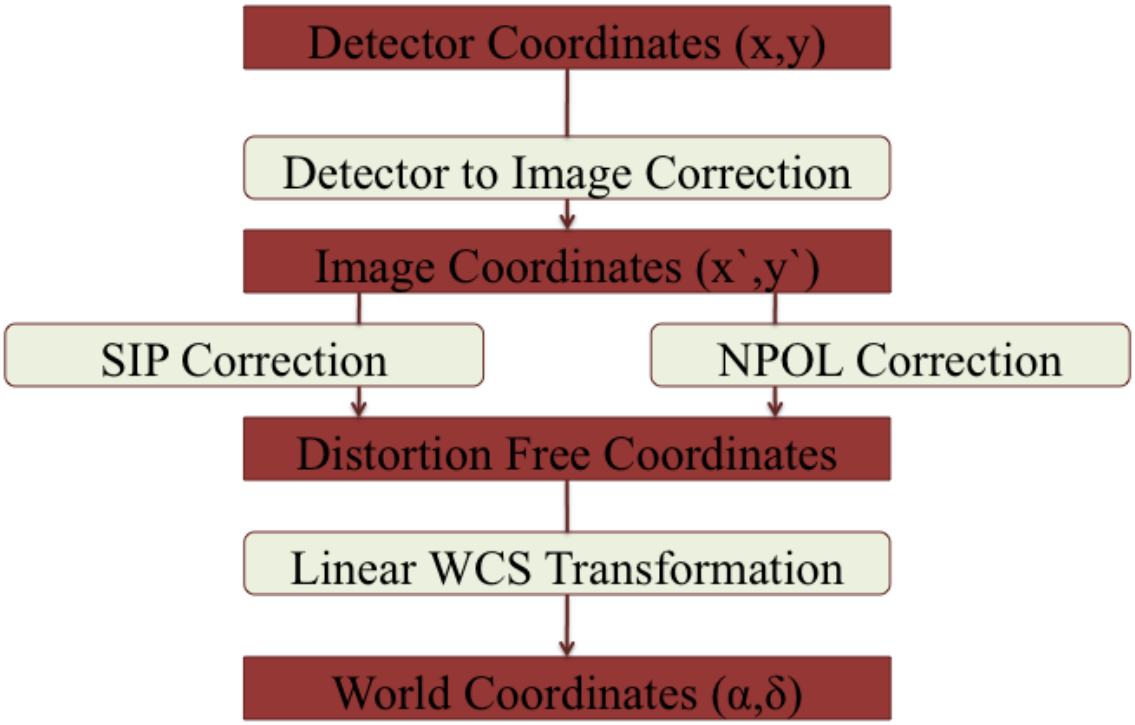


Fig. 3: Coordinate Transformation Pipeline

- read in velocity aberration correction factor (VAFACTOR) keyword
- apply velocity aberration, and the TDD terms for ACS data as well, to the distortion coefficients
  - write time-corrected distortion coefficients as the SIP keywords
- [if d2imfile is to be applied] read in D2IMFILE reference table
  - update D2IMEXT with name of reference table and AXISCORR keyword with axis to be corrected
  - append D2IMFILE array as a new `D2IMARR` extension
- [if NPOLFILE is to be applied] divide the NPOLFILE arrays by the linear distortion coefficients
  - write out normalized NPOLFILE arrays as new `WCSDVARR` extensions
  - update each SCI extension in the science image with the record-value keywords to point to the 2 `WCSDVARR` extensions (one for X corrections, one for Y corrections) associated with the SCI extension's chip

The STWCS task **updatewcs** applies these steps to update a science image's FITS file to incorporate the distortion model components using this convention. It not only modifies the input reference file data to apply to each image to account for time-dependent and velocity-aberration corrections as needed, but also creates the new extensions which get appended to the science image's FITS file.

Creating the D2IMARR extension

Converting the D2IMFILE reference table into a new `D2IMARR` FITS image extension involves only a few simple revisions to the header from D2IMFILE. The header of the `D2IMARR` extension consists of the following keywords required in order to properly interpret and apply the data in the extension to the science array:

- AXISCORR : Direction in which the det2im correction is applied
- EXTNAME : Set to 'D2IMARR'
- EXTVER : Set to 1
- NAXIS : Number of axes
- `NAXISj` : Size of each axis
- `CRPIXj` : Reference point for each axis, set at axis center
- `CRVALj` : computed from input science image array center on chip
- `CDELTj` : Binning of axis, computed as $1/BINAXIS_i$ keyword from science image

These keywords supplement the standard FITS required keywords for an image extension, including such keywords as PCOUNT, GCOUNT, BITPIX, and XTENSION.

The corrections specified in this extension refer to pixel positions on the detector. Since science images can be taken both as subarrays and in binned modes for some instruments, the subarray offset and binning factor get used to compute the translation from science image pixel position into unbinned full-detector pixel positions. Subarray exposures taken by HST detectors record the position of the detector's origin, (0,0) pixel, as `LTVj` keywords to identify what pixels on the physical detector were read out for the exposure. The conversion factor from image pixel position to physical detector pixel position of `(NAXISj/2 + LTVj)*BINAXISj` gets recorded as the `CRVALj` keyword value and gets used to correctly apply this correction to the science image.

In addition to the pixel position transformations encoded as the `D2IMARR` WCS, keywords reporting how the D2IM correction was created get copied into the new `D2IMARR` image extension header from the primary header of the D2IMFILE. This maintains as much provenance as possible for this correction.

A full listing of the `D2IMARR` extension for a sample ACS image can be found in *D2IMARR Header* in *Appendix 1: Headerlet API*.

Creating the WCSDVARR Extension

The NPOLFILE reference file contains at least 2 image extensions, one for the X correction and one for the Y correction for each chip. All these extensions get converted into their own WCSDVARR extension based on the FITS Distortion Paper convention when the NPOLFILE gets incorporated into the science image as another component of the distortion model. Both the array data for each NPOLFILE extension and the corresponding header needs to be modified before it can be written into the science image FITS file as a new WCSDVARR image extension.

The data from the NPOLFILE arrays represent the residuals after accounting for the distortion model, yet this correction gets applied as part of the distortion correction described in *Equation 4*. The linear terms of the distortion model need to be removed from the data in each NPOLFILE array in order to avoid applying the linear terms twice when applying the correction to the science data. This gets performed by reading in the linear distortion coefficients directly from the OCX and OCY keywords written out along with the SIP keywords, the multiplying them into the NPOLFILE data values using matrix dot operator to get the final, image specific NPOL correction to be written out as the WCSDVARR extension.

The header of this new WCSDVARR extension provides the translation from science image pixels to NPOLFILE array pixel positions as well as reporting on the provenance of the calibrations as recorded in the original NPOLFILE. The following keywords get computed based on the values directly from the NPOLFILE header:

- NAXISj : Length of each axis
- CDELTj : Step size in detector pixels along each axis for the NPOL array
- CRPIXj : Reference pixel position of NPOL array
- CRVALj : Reference pixel position of NPOL array relative to science array
- EXTNAME : always set to WCSDVARR
- EXTVER : identifier reported in the DP.EXTVER record-value keywords in the science array header

These keywords supplement the standard FITS required keywords for an image extension, including such keywords as PCOUNT, GCOUNT, BITPIX, and XTENSION. In addition, all keywords from the NPOLFILE primary header after and including 'FILENAME' get copied into the header of each WCSDARR extension to preserve the provenance of the calibration.

The look-up tables are saved as separate FITS image extensions in the science files with EXTNAME set to WCSDVARR. EXTVER is used when more than one look-up table is present in a single science file. Software which performs coordinate transformation will use bilinear interpolation to get the value of the distortion at a certain location in the image array. To fully map the image array to the distortion array the standard WCS keywords CRPIXj, CRVALj and CDELTj are used. The mapping follows the transformation

$$p_j = s_j(p_j - r_j) + w_j \tag{4.6}$$

where r_j is the CRPIXj value in the distortion array which corresponds to the w_j value in the image array, recorded as CRVALj in the WCSDVARR header. Elements in the distortion array are spaced by s_j pixels in the image array, where s_j is the CDELTj value in the distortion array header. In general s_j can have a non-integer value but cannot be zero. However, if the distortion array was obtained as a subimage of a larger array having a non-integer step size can produce undesirable results during interpolation. A full listing of the WCSDVARR extension for a sample ACS image can be found in *WCSDVARR Header* in *Appendix 1: Headerlet API*.

4.1.9 Summary

This paper describes a merging of previously proposed FITS WCS conventions to fully support the multi-component distortion models derived from calibrations for HST detectors. The application of this merged convention allows each science image to contain the full distortion model applicable to that specific image in an efficient and FITS compatible manner. The use of this calibration in the DrizzlePac package has been demonstrated to correct science data to much

better than 0.1 pixels across each image's field of view, with a typical RMS for aligning two ACS images on the order of 0.03 pixels in a suitably dense field of sources. This convention, despite making a few basic assumptions, retains each separate FITS convention's full functionality so that any software which understood, for example, the SIP standard will still work as before with the SIP keywords written out by the convention.

All HST ACS and WFC3 images retrieved from the archive have been updated using this convention so that users will no longer need to retrieve the distortion calibration data separately. Anyone using HST images will now be able to use the STWCS and/or DrizzlePac package to perform coordinate transformations or image alignment based on this convention, while still being able to use external tools like DS9 to take advantage of the SIP conventions as well. This solution now provides the best possible solution for supporting these highly accurate, yet complex multi-component distortion models in the most efficient manner available to data written out in the FITS format.

4.1.10 Appendix 1 - Sample ACS/WFC Image

The WCS of a single chip from an ACS/WFC exposure illustrates how the SIP keywords are populated based on the coefficients from the external IDCTAB reference file. In addition, this header includes the keywords referring to additional distortion corrections related to non-polynomial corrections from the NPOLFILE and to column-width corrections from the D2IMFILE. This sample illustrates how all three corrections can be specified at the same time in a FITS header using our rules for combining the SIP WCS convention and FITS Distortion Paper proposed syntax, while also using FITS WCS Paper I alternate WCS standards to maintain a record of the WCS information prior to being updated/recomputed to use the new reference information. The old WCS gets stored using WCS key 'O' and 'WCSNAMEO' = 'OPUS' to indicate it was originally computed by OPUS, the HST pipeline system.

FITS File extensions

The FITS file for this ACS/WFC image now contains extra extensions for the NPOLFILE and D2IMFILE corrections.

```
Filename: jbf401p8q_flc.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     261   ()           int16
1    SCI         ImageHDU       184   (4096, 2048)   float32
2    ERR         ImageHDU        55   (4096, 2048)   float32
3    DQ          ImageHDU        47   (4096, 2048)   int16
4    SCI         ImageHDU       183   (4096, 2048)   float32
5    ERR         ImageHDU        55   (4096, 2048)   float32
6    DQ          ImageHDU        47   (4096, 2048)   int16
7    D2IMARR     ImageHDU        12   (4096,)      float32
8    WCSDVARR    ImageHDU        37   (65, 33)     float32
9    WCSDVARR    ImageHDU        37   (65, 33)     float32
10   WCSDVARR    ImageHDU        37   (65, 33)     float32
11   WCSDVARR    ImageHDU        37   (65, 33)     float32
12   WCSCORR     BinTableHDU     59   14R x 24C    [40A, I, 1A, 24A, 24A, 24A, 24A, D,
↪ D, D, D, D, D, D, D,
                                                   24A, 24A, D, D, D, D, J, 40A,␣
↪128A]
```

The last extension, named WCSCORR, contains a binary table providing a summary of all the WCS solutions that have been applied to this file and does not act as an active part of the WCS or its interpretation.

Primary Header

The PRIMARY header of HST data contains keywords specifying information general to the entire file, such as what calibration steps were applied and what reference files should be used. No active WCS keywords (keywords interpreted for coordinate transformations) are present in the PRIMARY header, but keywords specifying the applicable distortion

reference files can be found in the PRIMARY header. Some keywords describing the distortion model and when the WCS was updated with the distortion information from the reference files can also be found in the PRIMARY header. These distortion and WCS related keywords from the PRIMARY header are:

```
             / CALIBRATION REFERENCE FILES

IDCTAB  = 'jref$v8q1444sj_idc.fits' / image distortion correction table
DGEOFILE= 'jref$qbu16420j_dxy.fits' / Distortion correction image
D2IMFILE= 'jref$v971826mj_d2i.fits' / Column Correction Reference File
NPOLFILE= 'jref$v971826aj_npl.fits' / Non-polynomial Offsets Reference File

UPWCSVER= '1.0.0   '           / Version of STWCS used to updated the WCS
PYWCSVER= '1.11-4.10'          / Version of PYWCS used to updated the WCS
DISTNAME= 'jbf401p8q_v8q1444sj-v971826aj-v971826mj'
SIPNAME = 'jbf401p8q_v8q1444sj'
```

The remainder of the PRIMARY header specifies the general characteristics of the image.

SCI Header Keywords

The following keywords only represent the WCS keywords from a sample ACS/WFC SCI header with 4-th order polynomial distortion correction from the IDCTAB reference file, along with NPOLFILE and D2IMFILE corrections from the specific reference files used as examples in *Appendix 2 - NPOLFILE Example Appendix 3 - D2IMFILE Example*.

```
XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32
NAXIS   =                    2
NAXIS1  =                 4096
NAXIS2  =                 2048
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
ORIGIN  = 'HSTIO/CFITSIO March 2010'
DATE    = '2012-06-13' / date this file was written (yyyy-mm-dd)
INHERIT =                    T / inherit the primary header
EXTNAME = 'SCI     '           / extension name
EXTVER  =                    1 / extension version number
ROOTNAME= 'jbf401p8q                        ' / rootname of the observation set
EXPNAME = 'jbf401p8q               ' / exposure identifier
BUNIT   = 'ELECTRONS'          / brightness units

             / WFC CCD CHIP IDENTIFICATION

CCDCHIP =                    2 / CCD chip (1 or 2)

             / World Coordinate System and Related Parameters

WCSAXES =                    2 / number of World Coordinate System axes
CRPIX1  =                 2048 / x-coordinate of reference pixel
CRPIX2  =                 1024 / y-coordinate of reference pixel
CRVAL1  =        11.3139376926 / first axis value at reference pixel
CRVAL2  =        42.0159325283 / second axis value at reference pixel
CTYPE1  = 'RA---TAN-SIP'       / the coordinate type for the first axis
CTYPE2  = 'DEC--TAN-SIP'       / the coordinate type for the second axis
CD1_1   = -7.8194868997837E-06 / partial of first axis coordinate w.r.t. x
CD1_2   = 1.09620231564470E-05 / partial of first axis coordinate w.r.t. y
```

(continues on next page)

(continued from previous page)

```
CD2_1   = 1.14279318521882E-05 / partial of second axis coordinate w.r.t. x
CD2_2   = 8.66885775536641E-06 / partial of second axis coordinate w.r.t. y
LTV1    =        0.0000000E+00 / offset in X to subsection start
LTV2    =        0.0000000E+00 / offset in Y to subsection start
LTM1_1  =                  1.0 / reciprocal of sampling rate in X
LTM2_2  =                  1.0 / reciprocal of sampling rate in Y
ORIENTAT=    51.66276166150634 / position angle of image y axis (deg. e of n)
RA_APER =   1.133205840898E+01 / RA of aperture reference position
DEC_APER=   4.202747924810E+01 / Declination of aperture reference position
PA_APER =              51.4653 / Position Angle of reference aperture center (de
VAFACTOR=   9.999374411935E-01 / velocity aberration plate scale factor

WCSCDATE= '18:41:12 (13/06/2012)' / Time WCS keywords were copied.
A_0_2   = 2.18045745103211E-06
B_0_2   = -7.2266555836441E-06
A_1_1   = -5.2225148886672E-06
B_1_1   = 6.20296011911662E-06
A_2_0   = 8.54842918202735E-06
B_2_0   = -1.7551668097547E-06
A_0_3   = 8.09354090167772E-12
B_0_3   = -4.2488740853874E-10
A_1_2   = -5.2903025382457E-10
B_1_2   = -7.6098727022982E-11
A_2_1   = -4.4821374838034E-11
B_2_1   = -5.1244088812978E-10
A_3_0   = -4.6755353102513E-10
B_3_0   = 8.48145748580355E-11
A_0_4   = -8.3665541956904E-17
B_0_4   = -2.1662072760964E-14
A_1_3   = -1.5108585176304E-14
B_1_3   = -1.5686763638364E-14
A_2_2   = 3.61252682019403E-14
B_2_2   = -2.6194214315839E-14
A_3_1   = 1.03502537140899E-14
B_3_1   = -2.6915637616404E-15
A_4_0   = 2.32643027828425E-14
B_4_0   = -1.5701287138447E-14
A_ORDER =                    4
B_ORDER =                    4
IDCSCALE=                 0.05
IDCV2REF=    256.6019897460938
IDCV3REF=    302.2520141601562
IDCTHETA=                  0.0
OCX10   = 0.001965125839177266
OCX11   =  0.04983026381230307
OCY10   =   0.0502766128737329
OCY11   = 0.001493971240339153
TDDALPHA=    0.246034678162242
TDDBETA = -0.07934489272074734
IDCXREF =               2048.0
IDCYREF =               1024.0
AXISCORR=                    1
D2IMEXT = '/grp/hst/cdbs/jref/v971826mj_d2i.fits'
D2IMERR = 0.002770500956103206
WCSNAMEO= 'OPUS    '
WCSAXESO=                    2
CRPIX1O =                 2048
```

(continues on next page)

(continued from previous page)

```
CRPIX2O =                1024
CDELT1O =                   1
CDELT2O =                   1
CUNIT1O = 'deg     '
CUNIT2O = 'deg     '
CTYPE1O = 'RA---TAN-SIP'
CTYPE2O = 'DEC--TAN-SIP'
CRVAL1O =       11.3139376926
CRVAL2O =       42.0159325283
LONPOLEO=                 180
LATPOLEO=       42.0159325283
RESTFRQO=                   0
RESTWAVO=                   0
CD1_1O  =   -7.81948731152E-06
CD1_2O  =    1.09620228331E-05
CD2_1O  =    1.14279315609E-05
CD2_2O  =    8.66885813904E-06
WCSNAME = 'IDC_v8q1444sj'
CPERR1  =                 0.0 / Maximum error of NPOL correction for axis 1
CPDIS1  = 'Lookup  '            / Prior distortion funcion type
DP1     = 'EXTVER: 1' / Version number of WCSDVARR extension
DP1     = 'NAXES: 2' / Number of independent variables in CPDIS function
DP1     = 'AXIS.1: 1' / Axis number of the 1st variable in a CPDIS function
DP1     = 'AXIS.2: 2' / Axis number of the 2nd variable in a CPDIS function
CPERR2  =                 0.0 / Maximum error of NPOL correction for axis 2
CPDIS2  = 'Lookup  '            / Prior distortion funcion type
DP2     = 'EXTVER: 2' / Version number of WCSDVARR extension
DP2     = 'NAXES: 2' / Number of independent variables in CPDIS function
DP2     = 'AXIS.1: 1' / Axis number of the 1st variable in a CPDIS function
DP2     = 'AXIS.2: 2' / Axis number of the 2nd variable in a CPDIS function
NPOLEXT = 'jref$v971826aj_npl.fits'
```

All keywords related to the exposure itself, such as readout pattern, have been deleted from this SCI header listing for the sake of brevity.

D2IMARR Header

The full, complete header of the `D2IMARR` extension as derived from the D2IMFILE discussed in *Appendix 3 - D2IMFILE Example*.

```
XTENSION= 'IMAGE   '            / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    1 / number of array dimensions
NAXIS1  =                 4096
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
AXISCORR=                    1 / Direction in which the det2im correction is app
EXTVER  =                    1 / Distortion array version number
EXTNAME = 'D2IMARR '            / WCS distortion array
CDELT1  =                  1.0 / Coordinate increment along axis
CRPIX1  =               2048.0 / Coordinate system reference pixel
CRVAL1  =               2048.0 / Coordinate system value at reference pixel
```

WCSDVARR Header

Each of the WCSDVARR extensions has been derived based on the values for the NPOL correction found in the reference file described in *Appendix 2 - NPOLFILE Example*. The full header for the WCSDVARR extension with EXTVER=1 is:

```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   65
NAXIS2  =                   33
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTVER  =                    1 / Distortion array version number
EXTNAME = 'WCSDVARR'           / WCS distortion array
CRVAL2  =                  0.0 / Coordinate system value at reference pixel
CRPIX1  =                  0.0 / Coordinate system reference pixel
CRPIX2  =                  0.0 / Coordinate system reference pixel
CRVAL1  =                  0.0 / Coordinate system value at reference pixel
CDELT1  =                   64 / Coordinate increment along axis
CDELT2  =                   64 / Coordinate increment along axis
FILENAME= 'v971826aj_npl.fits' / name of file
FILETYPE= 'DXY GRID'           / type of data found in data file
OBSTYPE = 'IMAGING '           / type of observation
TELESCOP= 'HST'                / telescope used to acquire data
INSTRUME= 'ACS   '             / identifier for instrument used to acquire data
DETECTOR= 'WFC'                / detector in use: WFC, HRC, or SBC
FILTER1 = 'F475W   '           / element selected from filter wheel 1
FILTER2 = 'CLEAR2L '           / element selected from filter wheel 2
USEAFTER= 'Mar 01 2002 00:00:00'
COMMENT = 'NPOL calibration file created by Ray A. Lucas 29 APR 2010'
DESCRIP = 'Residual geometric distortion file for use with astrodrizzle-------'
PEDIGREE= 'INFLIGHT 11/11/2002 11/11/2002'
HISTORY   Non-polynomial offset file generated from qbu16420j_dxy.fits
HISTORY   Only added to the flt.fits file and used in coordinate
HISTORY   transformations if the npol reference filename is specified in
HISTORY   the header.  The offsets are copied from the reference file into
HISTORY   two arrays for each chip.  Each array is stored as a 65x33 pixel
HISTORY   image that gets interpolated up to the full chip size. Two new
HISTORY   extensions for each chip are also appended to the flt file
HISTORY   (WCSDVARR).
HISTORY qbu16420j_npl.fits renamed to v9615069j_npl.fits on Sep 6 2011
HISTORY v9615069j_npl.fits renamed to v971826aj_npl.fits on Sep 7 2011
```

Each of the `WCSDVARR` extension headers contains the same set of keywords, with only the values varying to reflect the axis and chip corrected by this extension.

4.1.11 Appendix 2 - NPOLFILE Example

The NPOLFILE reference file format includes a PRIMARY header describing what kind of image should be corrected by this file, along with extensions containing the corrections for each chip.

FITS File Extensions

A sample NPOLFILE applicable to ACS/WFC F475W images has the FITS extensions:

```
Filename: /grp/hst/cdbs/jref/v971826aj_npl.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      35   ()           int16
1    DX          ImageHDU       180   (65, 33)     float32
2    DY          ImageHDU       215   (65, 33)     float32
3    DX          ImageHDU       215   (65, 33)     float32
4    DY          ImageHDU       215   (65, 33)     float32
```

The extensions with the name 'DX' provide the corrections in X for each of the ACS/WFC's 2 chips, while the 'DY' extensions provide the corrections in Y for each chip.

Primary Header

The PRIMARY header of this file only includes the minimum information necessary to describe what exposures should be corrected by this reference file and how it was generated. A full listing of the PRIMARY header includes:

```
SIMPLE  =                    T / Fits standard
BITPIX  =                   16 / Bits per pixel
NAXIS   =                    0 / Number of axes
EXTEND  =                    T / File may contain extensions
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
IRAF-TLM= '2011-09-09T13:24:40'
NEXTEND =                    4 / Number of standard extensions
DATE    = '2010-04-02T19:53:08'
FILENAME= 'v971826aj_npl.fits' / name of file
FILETYPE= 'DXY GRID'           / type of data found in data file
OBSTYPE = 'IMAGING '           / type of observation
TELESCOP= 'HST'                / telescope used to acquire data
INSTRUME= 'ACS   '             / identifier for instrument used to acquire data
DETECTOR= 'WFC'                / detector in use: WFC, HRC, or SBC
FILTER1 = 'F475W   '           / element selected from filter wheel 1
FILTER2 = 'CLEAR2L '           / element selected from filter wheel 2
USEAFTER= 'Mar 01 2002 00:00:00'
COMMENT = 'NPOL calibration file created by Ray A. Lucas 29 APR 2010'
DESCRIP = 'Residual geometric distortion file for use with astrodrizzle-------'
PEDIGREE= 'INFLIGHT 11/11/2002 11/11/2002'
HISTORY   Non-polynomial offset file generated from qbu16420j_dxy.fits
HISTORY   Only added to the flt.fits file and used in coordinate
HISTORY   transformations if the npol reference filename is specified in
HISTORY   the header.  The offsets are copied from the reference file into
HISTORY   two arrays for each chip.  Each array is stored as a 65x33 pixel
HISTORY   image that gets interpolated up to the full chip size. Two new
HISTORY   extensions for each chip are also appended to the flt file
HISTORY   (WCSDVARR).
HISTORY qbu16420j_npl.fits renamed to v9615069j_npl.fits on Sep 6 2011
HISTORY v9615069j_npl.fits renamed to v971826aj_npl.fits on Sep 7 2011
```

Data Extension Header

Each ACS/WFC chip has a shape of 4096 x 2048 pixels, yet the data arrays in this specific reference file only have 65x33 values. Each data extension ('DX' and 'DY') contains only those keywords necessary to properly interpolate the sub-sampled values from the arrays to apply to each individual pixel in the full ACS/WFC exposure. The full header for the ['DX',1] extension contains:

```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / Bits per pixel
NAXIS   =                    2 / Number of axes
NAXIS1  =                   65 / Axis length
NAXIS2  =                   33 / Axis length
PCOUNT  =                    0 / No 'random' parameters
GCOUNT  =                    1 / Only one group
EXTNAME = 'DX      '           / Extension name
EXTVER  =                    1 / Extension version
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
INHERIT =                    F / Inherits global header
DATE    = '2004-04-28T16:44:21'
IRAF-TLM= '16:42:00 (30/11/2006)'
WCSDIM  =                    2
LTM1_1  =                   1.
LTM2_2  =                   1.
WAT0_001= 'system=physical'
WAT1_001= 'wtype=linear'
WAT2_001= 'wtype=linear'
CCDCHIP =                    2 / CCDCHIP from full size dgeo file
LTV1    =                    0
LTV2    =                    0
ONAXIS1 =                 4096 / NAXIS1 of full size dgeo file
ONAXIS2 =                 2048 / NAXIS2 of full size dgeo file
CDELT1  =                   64 / Coordinate increment along axis
CDELT2  =                   64 / Coordinate increment along axis
```

4.1.12 Appendix 3 - D2IMFILE Example

The D2IMFILE reference file only contains a single 1-D array that should correct the column (row) values based on the value of the 'AXISCORR' keyword in the SCI header.

FITS File Extensions

This simple reference file, therefore, contains only 2 extensions; namely,

```
Filename: /grp/hst/cdbs/jref/v971826mj_d2i.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      35   ()           int16
1    DX          ImageHDU        18   (4096,)      float32
```

PRIMARY Header

The PRIMARY header only needs to contain information on what detector this file corrects, along with any available information on how this file was generated. The ACS/WFC D2IMFILE PRIMARY header only includes:

```
SIMPLE  =                    T / Fits standard
BITPIX  =                   16 / Bits per pixel
NAXIS   =                    0 / Number of axes
EXTEND  =                    T / File may contain extensions
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
DATE    = '2010-02-01T20:19:11' / Date FITS file was generated
IRAF-TLM= '2011-09-02T13:04:07' / Time of last modification
```

(continues on next page)

(continued from previous page)

```
NEXTEND =                    1 / number of extensions in file
FILENAME= 'v971826mj_d2i.fits' / name of file
FILETYPE= 'WFC D2I FILE'         / type of data found in data file
OBSTYPE = 'IMAGING '             / type of observation
TELESCOP= 'HST'                / telescope used to acquire data
INSTRUME= 'ACS   '             / identifier for instrument used to acquire data
DETECTOR= 'WFC     '
USEAFTER= 'Mar 01 2002 00:00:00'
COMMENT = 'D2I calibration file created by Warren Hack 29 APR 2010'
DESCRIP = 'Column-width correction file for WFC images-----------------------'
PEDIGREE= 'INFLIGHT 11/11/2002 11/11/2002'
HISTORY
HISTORY   Fixed column (or row) width correction file. This is applied
HISTORY   as a correction to the input pixel position and the output of
HISTORY   this correction is to be used as input to the polynomial and
HISTORY   non-polynomial distortion corrections.
HISTORY
HISTORY   For ACS WFC data, the correction is stored as an image extension
HISTORY   (D2IMARR) with one row. Each element in the row specifies the
HISTORY   correction in pixels for every pixel in the column (or row) in
HISTORY   the science extension; for ACS WFC, the correction is in the X
HISTORY   direction.
HISTORY
HISTORY   For a more in-depth explanation of this file, please see the
HISTORY   draft writeup at:
HISTORY http://stsdas.stsci.edu/stsci_python_epydoc/stwcs/fits_conventions.html
HISTORY wfc_ref68col_d2i.fits renamed to v961506lj_d2i.fits on Sep 6 2011
HISTORY v961506lj_d2i.fits renamed to v971826mj_d2i.fits on Sep 7 2011
```

In this case, most of the keywords not required by FITS describe how this file was computed while also describing how it should be applied.

Data Extension Header

The header keywords for the actual DX array simply needs to provide the information necessary to apply the values to the data; namely,

```
XTENSION= 'IMAGE   '           / Image extension
BITPIX  =                  -32 / Bits per pixel
NAXIS   =                    1 / Number of axes
NAXIS1  =                 4096 / Axis length
PCOUNT  =                    0 / No 'random' parameters
GCOUNT  =                    1 / Only one group
EXTNAME = 'DX      '           / Extension name
EXTVER  =                   11 / Extension version
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
INHERIT =                    F / Inherits global header
DATE    = '2009-03-18T19:28:09' / Date FITS file was generated
IRAF-TLM= '16:05:02 (18/03/2009)' / Time of last modification
CRPIX1  =                    0 / Distortion array reference pixel
CDELT1  =                    0 / Grid step size in first coordinate
CRVAL1  =                    0 / Image array pixel coordinate
CRPIX2  =                    0 / Distortion array reference pixel
CDELT2  =                    0 / Grid step size in second coordinate
CRVAL2  =                    0 / Image array pixel coordinate
```

The fact that these values get applied without interpolation to each pixel in a row, in this case, means that no translation terms are needed in the header, making for a very simple header and very simple application to the data.

4.2 TSR 2012-03: NPOL Reference File

Abstract: Authors: Nadezhda Dencheva, Warren Hack Date: 12 Oct 2010

The HST pipeline originally used two types of reference files to correct for distortion: IDCTAB files which contain the coefficients for a polynomial correction and DGEO files which are images with the residual distortion. A new format for the residual distortion, called NPOL files (extension _npl.fits), is presented in this document. The conversion from DGEO files to NPOL files is described and an example of the format is given using ACS/WFC F606W filter. Tests of the conversion procedure show that the differences between the DGEO files and NPOL files are of the order of 10^{-4} pixels.

Contents:

4.2.1 NPOL Reference File

Introduction

HST images can exhibit significant distortion, one of the severe cases being ACS/WFC where it can reach 50 pixels. Anderson [Anderson2002] describes the total distortion solution for ACS/WFC as consisting of a polynomial part which provides position accuracy of 0.1-0.2 pixels, a filter dependent fine scale solution which brings the accuracy of the positions to 0.01 pixels and a detector defect correction with a maximum amplitude of 0.008 pixels. These distortion solutions are implemented in the ACS pipeline as reference files. The IDCTAB files contain the polynomial distortion and the DGEO files originally contained the combined solution for the detector defect and the filter dependent fine scale residuals. This document describes how the DGEO files are converted to the new format, called NPOL files, and how they will be distributed and used. It also describes the testing procedure of the NPOL files and provides an example of converting and testing an ACS/WFC F606W DGEO file.

New representation - look-up tables

The fine scale distortions represented in the DGEO images can be stored in smaller look-up tables without significant loss of information. These look-up tables follow the conventions in the WCS FITS Distortion Paper [Calabretta2004]. Record-valued keywords are used to map an image in the science extension to a distortion array in the WCSDVAR extension. This new type of FITS keywords has been implemented in PyFITS and is fully described in [Calabretta2004]. Specifically, DPj.EXTVER in the science extension header maps the science image to the correct WCSDVAR extension. The dimensionality of the distortion array is defined by DPj.NAXES. Keywords DPj.AXIS.j in the SCI extension header are used for mapping image array axis to distortion array axis. In the keywords above j is an integer and denotes the axis number. For example, if distortion array axis 1 corresponds to image array axis 1 of a SCI extension, then DP.1.AXIS.1 = 1. A full example of the keywords added to a SCI extension header is presented in the last section.

The look-up tables are saved as separate FITS image extensions in the science files with EXTNAME set to WCSDVARR. EXTVER is used when more than one look-up table is present in a single science file. Software which performs coordinate transformation will use bilinear interpolation to get the value of the distortion at a certain location in the image array. To fully map the image array to the distortion array the standard WCS keywords CRPIXj, CRVALj and CDELTj are used. The mapping follows the transformation

$$p_j = s_j(p_j - r_j) + w_j$$

where r_j is the CRPIXj value in the distortion array which corresponds to the w_j value in the image array, recorded as CRVALj in the WCSDVARR header. Elements in the distortion array are spaced by s_j pixels in the image array,

where s_j is the CDELTj value in the distortion array header. In general s_j can have a non-integer value but cannot be zero. However, if the distortion array was obtained as a subimage of a larger array having a non-integer step size can produce undesirable results during interpolation. An example header for ACS/WFC F606W WCSDVARR extension header is given in the last section.

A note about look-up tables

It is essential that the look-up tables meet two restrictions:

- Every point in the corrected image is mapped to by not more than one point in the uncorrected image.
- Every point in the corrected image is mapped to by at least one point on the corrected image.

This one-to-one (non-extrapolation) requirement can have implications on the geometry of the distortion array. If the distortion array is obtained as a subimage or subsample of a larger array, it is important that the edges of the distortion array coincide with the edges of the image.

Creating an NPOL file from a DGEO file

The DGEO files are FITS files with four image extensions with full chip size 4096x2048 pixels representing the residuals of the distortion in X and Y for the two ACS/WFC chips. As described by Anderson [Anderson2002], the original tables from which the full size DGEO images were created were sampled every 64 pixels to a size of 65x33 pixels. Because of the coordinate transformations and many steps involved in creating the DGEO files it was not possible to start with the original tables. Our purpose was to sample the full size DGEO files in such a way that after interpolating them again the newly expanded images would match the original images as close as possible. This is why we chose a step size of 64 pixels for the sampling. Given the non-extrapolation restriction and the requirement to have an integer step size we needed to sample an image of a size 4097x2049. We copied the last row/column of the DGEO images to the extra row/column before sampling. This padding ensures that after bilinear interpolation there all edge effects due to extrapolation will be minimized.

A Python script, makesmall.py, samples the large DGEO files and writes out the small NPOL files. This code has been included in the REFTOOLS package in the stsci_python distribution. The script also writes the sampling step size in each direction to the headers of the NPOL file extensions. The step size is later stored in the header of each WCSDVAR extension as the value of CDELT keywords to be used by the software which does the coordinate transformation and interpolation. Since the original DGEO files include the combined fine scale distortion and the detector defect, it is imperative that the detector defect is removed from the DGEO files before they are sampled. (The detector defect correction is stored also as a D2IMARR extension and applied separately.)

Using NPOL files

STWCS.UPDATEWCS is used to incorporate all available distortion information for a given observation in the science file. The name of the NPOL file which stores the residual distortion for a specific science observation is written in the NPOLFILE keyword in the primary header. UPDATEWCS copies the NPOL file extensions as WCSDVARR extensions in the science file. The header of each WCSDVARR extension is also created at this time following the rules in section 2 and the necessary record-valued keywords are inserted in the science extension header so that the axes in the science image are mapped to the correct WCSDVARR extension.

STWCS.WCSUTIL and its main class HSTWCS, as well as its base class PyWCS.WCS, can read and interpret FITS files with WCSDVARR extensions. The method which performs the bilinear interpolation and corrects the coordinates is p4_pix2foc(). All coordinate transformation methods distinguish between 0-based and 1-based input coordinates through the origin parameter.

A note about the fine scale distortion:

The original fine scale distortion was meant to be applied after the polynomial IDCTAB distortion. In the new coordinate transformation pipeline the polynomial distortion follows the SIP convention and the first order coefficients are incorporated in the CD matrix which is used last in the pipeline to transform from distortion corrected coordinates to sky coordinates. As a consequence residual distortion arrays must be corrected with the inverse of the CD matrix since they will be applied before the first order coefficients. UPDATEWCS performs this correction for each extension of the NPOL file. However, when we test the NPOL files this correction is omitted because the test does not require performing the entire coordinate transformation pipeline from detector to sky coordinates.

Testing NPOL files

A Python script, REFTOOLS.test_small_dgeo.py, was written and made available for testing of the NPOL files. The following procedure is implemented in the test script:

- A science observation is run through STWCS.UPDATEWCS to update the headers and create the WCSDVAR extensions.
- An HSTWCS object is created from a SCI extension
- A regular grid with the size of the image is created and is passed as input to
  - the HSTWCS.det2im() method to account for the column correction reported in the D2IM reference file, then
  - to the HSTWCS.p4_pix2foc() method which applies bilinear interpolation to the WCSDVARR extension to the input grid.
- The expanded NPOL file is compared to the original full size DGEO file and the difference images are (optionally) written to a file.

This comparison allows us to verify that the NPOL files get interpolated to produce the exact same correction as provided by the DGEO files for the same pixel position. Any further comparisons based on the full coordinate transformation with and without these corrections get masked by the differences in how the input FLT image coordinates get transformed to pixel positions in the output image.

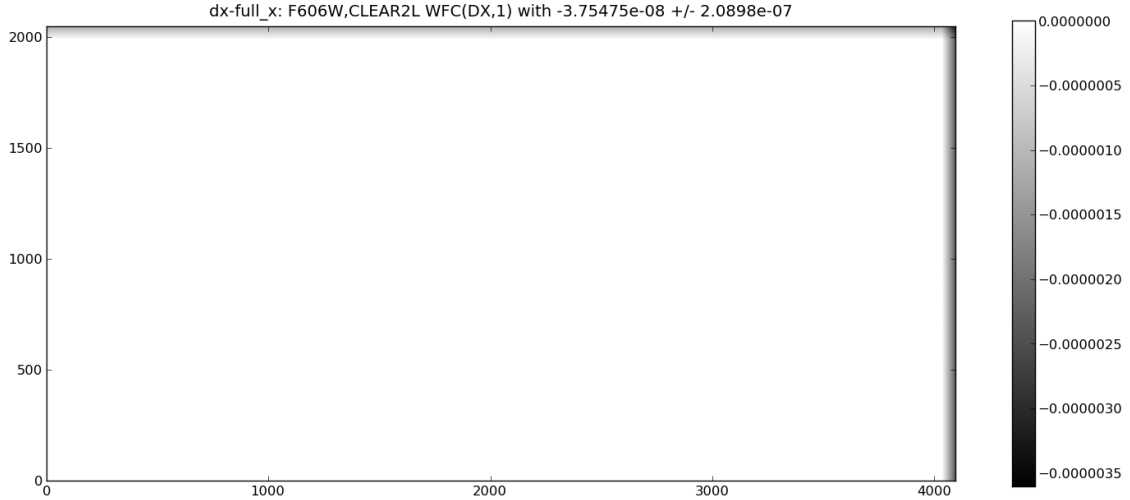
Results

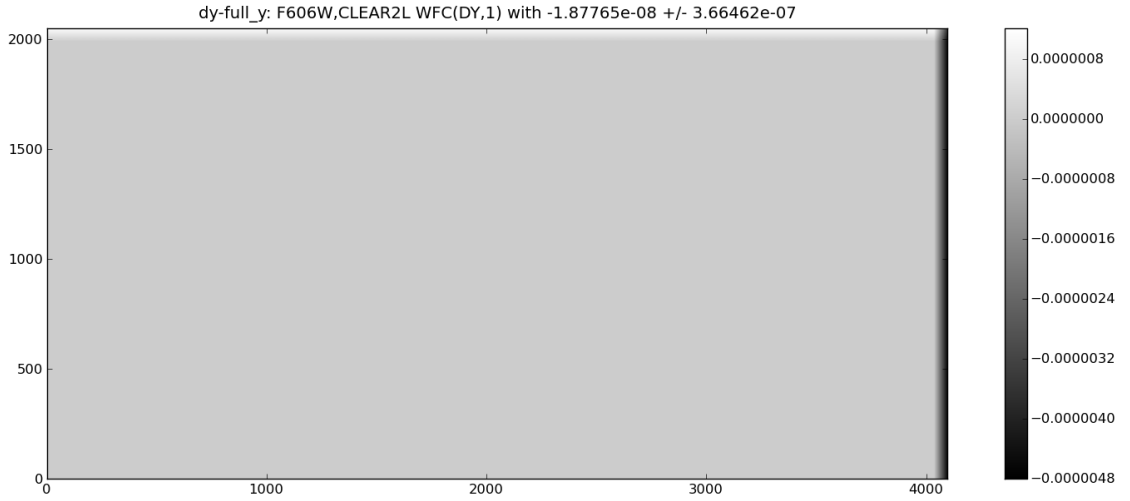
The best way to verify that the transformation from sub-sampled NPOLFILE into the full-frame represented by the full-size DGEOFILE was to use an artificial DGEOFILE. This artificial DGEOFILE consisted of a strictly bilinear plane in the DX and DY arrays. This should be something that the bilinear interpolation routines in STWCS/PyWCS can exactly match when expanding the NPOLFILE, which was created by sub-sampling the full-size DGEOFILE. This also allows us to verify that we know how to specify the header for the NPOLFILE extensions as written out to the FLT images to insure that the proper expansion gets performed by STWCS/PyWCS.

The residuals from this comparison came out to be within single-point floating point precision with the exception of the edge effects in the last few rows and columns of the expanded array as seen here:

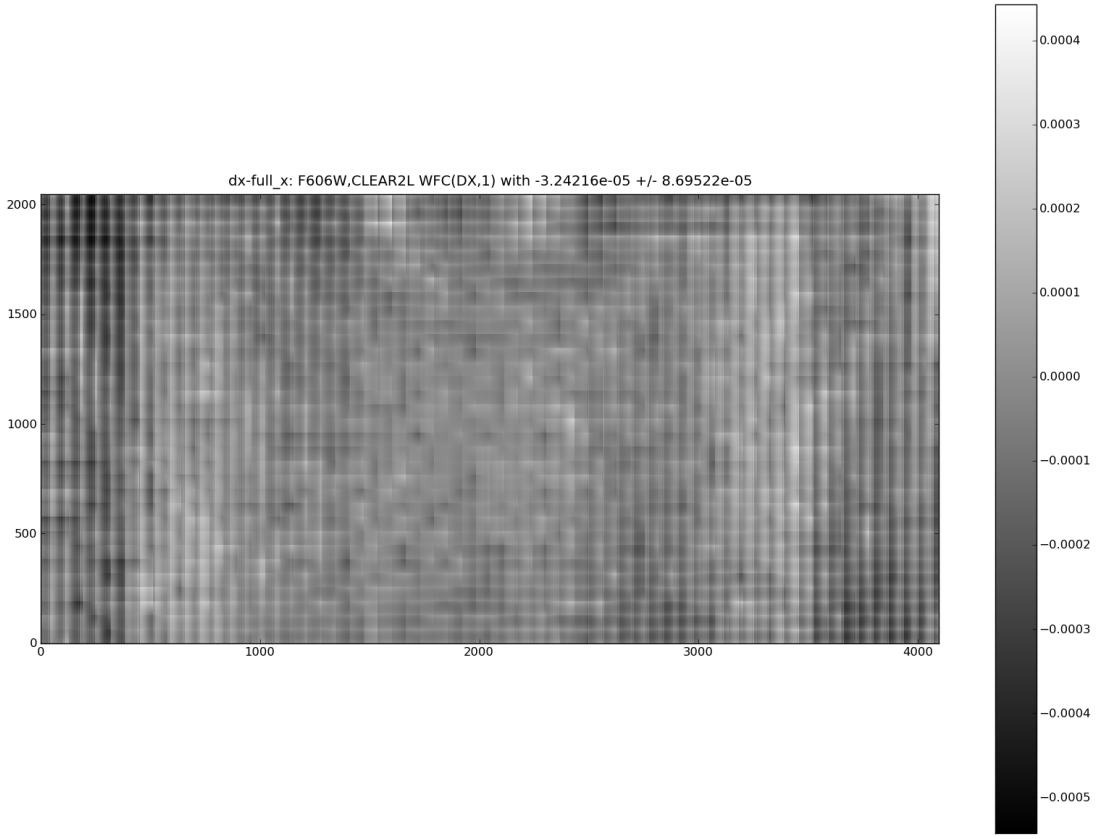
This test confirmed that the interpolation routine implemented within PYWCS will correctly expand the NPOL file points to exactly recreate the DGEO file correction for any given pixel position, except at the far ends of the columns of rows. The variations at the ends of the rows and tops of the columns comes from edge effects of the interpolation as it interpolates over 1 less pixel at the edges, however, even these variations are well within numerical accuracy for the overall correction.

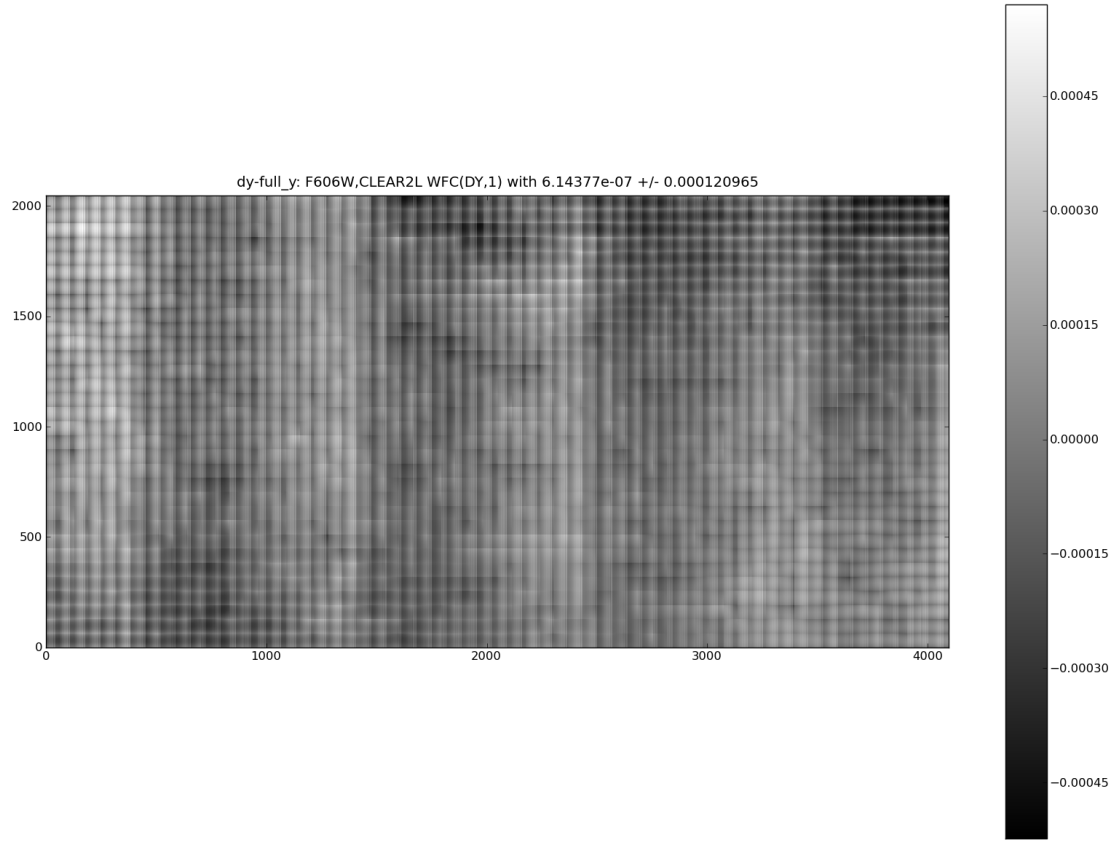
The new NPOL reference files were then compared to actual DGEO files from CDBS for an ACS/WFC F606W image using this testing code. The test image was run through STWCS.UPDATEWCS to populate the headers and write the

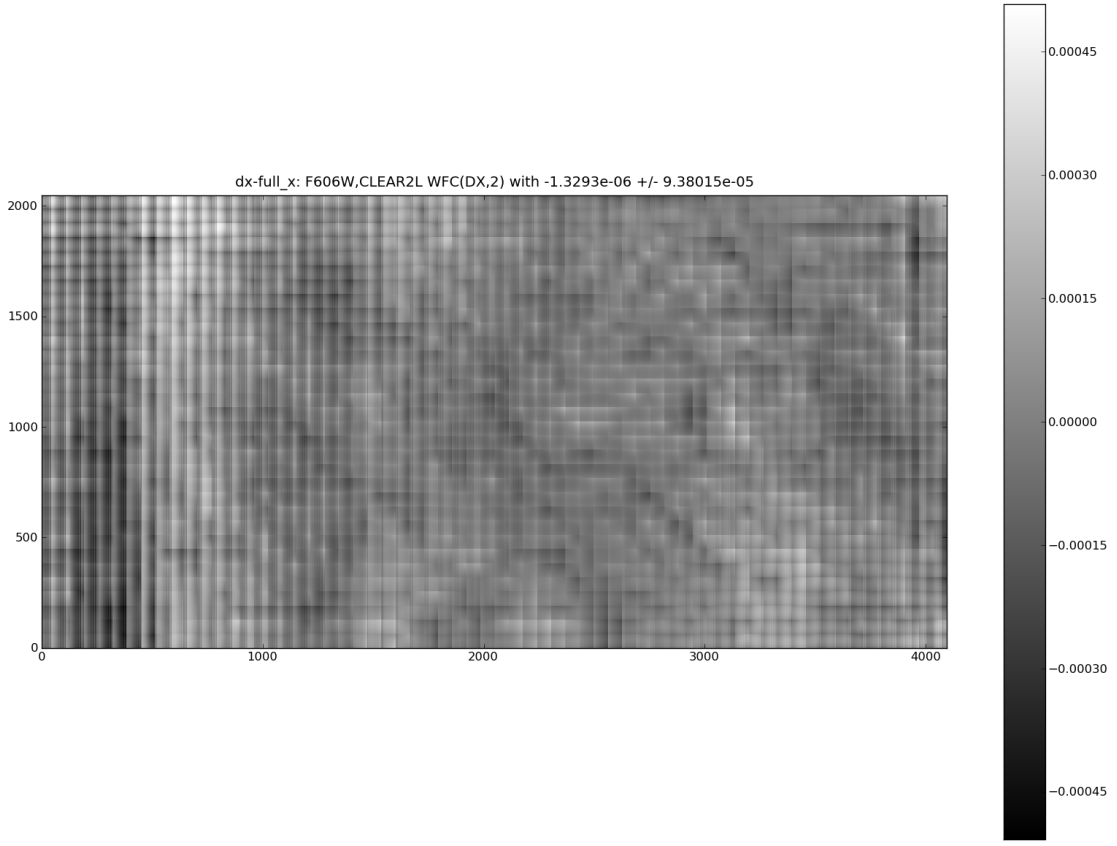

dx-full_x: F606W,CLEAR2L WFC(DX,1) with -3.75475e-08 +/- 2.0898e-07
2000
1500
1000
500
0
0
1000
2000
3000
4000
0.0000000
−0.0000005
−0.0000010
−0.0000015
−0.0000020
−0.0000025
−0.0000030
−0.0000035

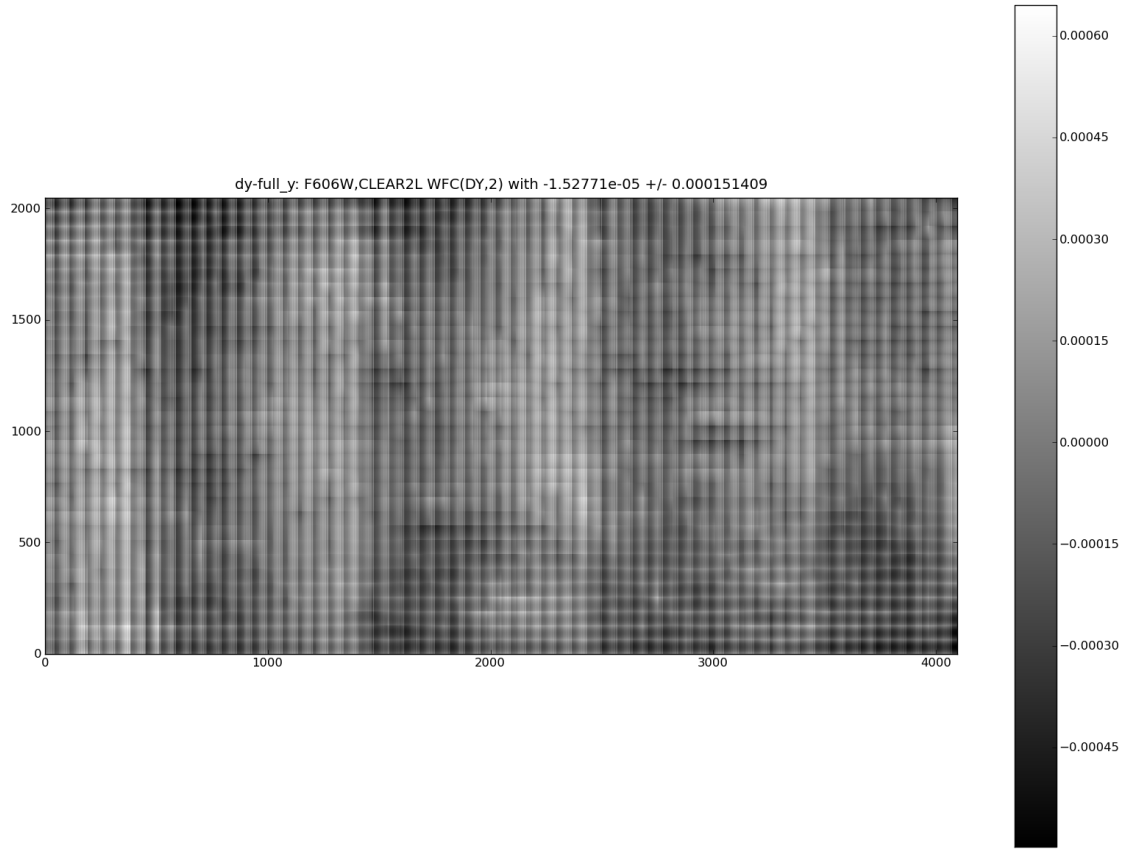

dy-full_y: F606W,CLEAR2L WFC(DY,1) with -1.87765e-08 +/- 3.66462e-07
2000
1500
1000
500
0
0
1000
2000
3000
4000
0.0000008
0.0000000
−0.0000008
−0.0000016
−0.0000024
−0.0000032
−0.0000040
−0.0000048

WCSDVAR extensions. Fig 3-6 show the difference between the DGEO files and the expanded NPOL files for the two ACS/WFC chips in X and Y.

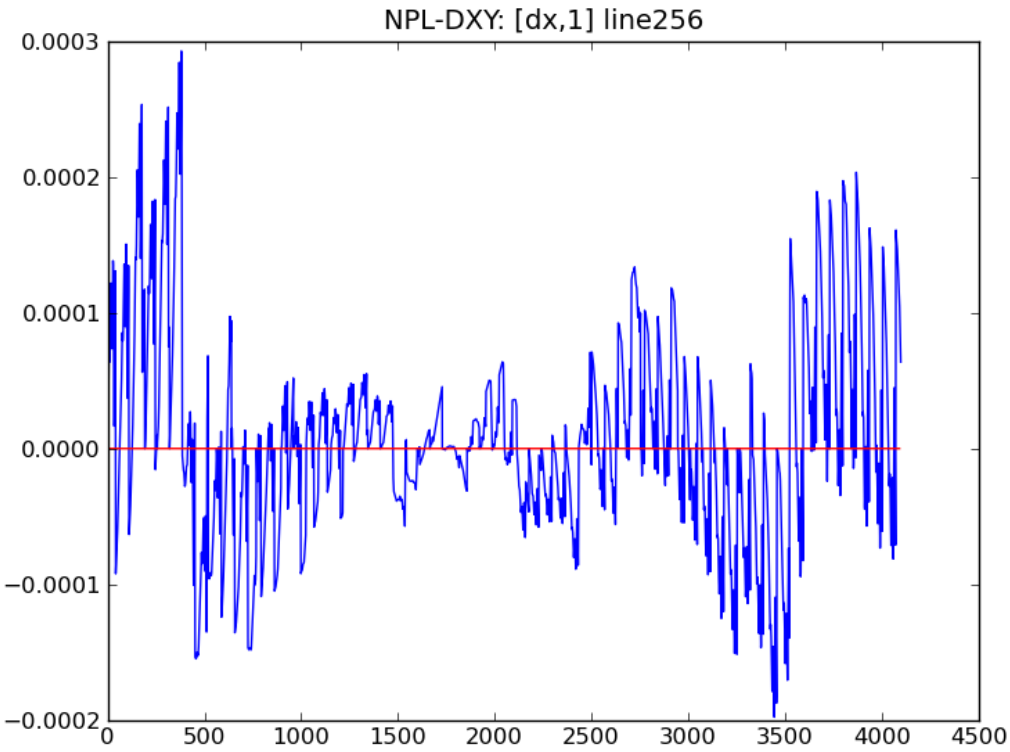


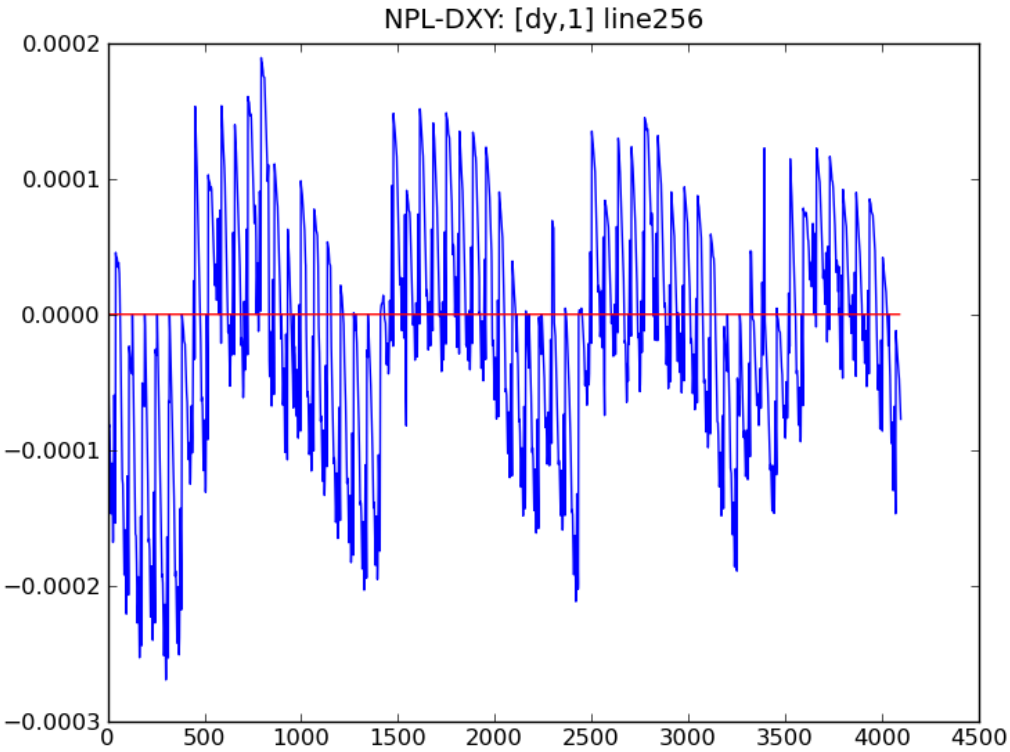

dy-full_y: F606W,CLEAR2L WFC(DY,1) with 6.14377e-07 +/- 0.000120965
2000
1500
1000
500
0
0
1000
2000
3000
4000
0.00045
0.00030
0.00015
0.00000
−0.00015
−0.00030
−0.00045


dx-full_x: F606W,CLEAR2L WFC(DX,2) with -1.3293e-06 +/- 9.38015e-05
2000
1500
1000
500
0
0
1000
2000
3000
4000
0.00045
0.00030
0.00015
0.00000
−0.00015
−0.00030
−0.00045



A random line from the difference image in X and Y is shown in the next two plots.


NPL-DXY: [dx,1] line256
0.0003
0.0002
0.0001
0.0000
−0.0001
−0.0002
0
500
1000
1500
2000
2500
3000
3500
4000
4500


NPL-DXY: [dy,1] line256
0.0002
0.0001
0.0000
−0.0001
−0.0002
−0.0003
0
500
1000
1500
2000
2500
3000
3500
4000
4500

These results were used as the intial indication that the NPOL lookup tables accurately reproduce the same corrections as the original full-size DGEO reference images while avoiding the confusion of a full coordinate transformation. Further testing by the ACS Instrument Team will independently confirm whether or not the code and the new reference files accurately correct ACS images before these new reference files will be made available for general use or even for use in the pipeline.

4.2.2 Appendix

Sample WCSDVARR extension header:

```
XTENSION= 'IMAGE   '            / Image extension
BITPIX  =                   -32 / array data type
NAXIS   =                     2 / number of array dimensions
NAXIS1  =                    65
NAXIS2  =                    33
PCOUNT  =                     0 / number of parameters
GCOUNT  =                     1 / number of groups
EXTVER  =                     1 / Distortion array version number
EXTNAME = 'WCSDVARR'            / WCS distortion array
CRVAL2  =                   0.0 / Coordinate system value at reference pixel
CRPIX1  =                   0.0 / Coordinate system reference pixel
CRPIX2  =                   0.0 / Coordinate system reference pixel
CRVAL1  =                   0.0 / Coordinate system value at reference pixel
CDELT1  =                    64 / Coordinate increment along axis
CDELT2  =                    64 / Coordinate increment along axis
```

Sample science extension keywords defining a lookup table distortions:

```
CPERROR1=                0.058602706   / Maximum error of dgeo correction for axis 1
CPDIS1  = 'Lookup  '            / Prior distortion function type
DP1     = 'EXTVER: 1.0'         / Version number of WCSDVARR extension
DP1     = 'NAXES: 2.0'          / Number of independent variables in CPDIS function
DP1     = 'AXIS.1: 1.0'         / Axis number of the 1st variable in a CPDIS function
DP1     = 'AXIS.2: 2.0'         / Axis number of the 2nd variable in a CPDIS function
CPERROR2=         0.072911568  / Maximum error of dgeo correction for axis 2
CPDIS2  = 'Lookup  '            / Prior distortion function type
DP2     = 'EXTVER: 2.0'         / Version number of WCSDVARR extension
DP2     = 'NAXES: 2.0'          / Number of independent variables in CPDIS function
DP2     = 'AXIS.1: 1.0'         / Axis number of the 1st variable in a CPDIS function
DP2     = 'AXIS.2: 2.0'         / Axis number of the 2nd variable in a CPDIS function
```

4.3 TSR 2012-02: Definition of a Headerlet and Its Role in Updating WCS Information

Abstract:: Authors: Warren Hack, Nadezhda Dencheva Date: 22 Oct 2012

A headerlet is a self-consistent representation of a single WCS solution for a single exposure complete with all distortion information. FITS is the data storage format currently supported. It has no observational data which makes it relatively small and light to distribute. It is, essentially, a mechanism for encapsulating WCS information which can later be used to update the WCS of a science file and allows improved astrometric solutions to be stored and passed around easily. The HST archive is expected to start accepting headerlets for HST data soon. However the implementaiton is not HST specific and FITS WCS standard, as well as all WCS conventions implemented in pywcs are supported. This report describes the format

and contents of a headerlet along with the software implementation and methods for creating headerlets and using them to update the WCS of a science observation.

Contents:

4.3.1 Introduction

The original motivation for this work was a WCS based replacement of Multidrizzle, now released as Astrodrizzle, and specifically a requirement for the availability and management of multiple WCS sets, complete with distortion, within one science file. However, the concept of encapsulating astrometric solutions is more general than that since each solution may represent a different astrometric alignment, either with a catalog or another image. Furthermore, computing accurate astrometric solutions requires considerable effort and time so having a way to distribute them, apply them to a science observation and switch between different WCSs efficiently would facilitate many aspects of data analysis.

Some of the immediate areas for the use of headerlets with HST data include the HST archive, the HLA and other legacy projects which provide improved astrometry of HST observations. The HST Archive, for example, provides access to all HST data, most of which can not be readily combined together due to errors in guide star astrometry imposing offsets between images taken using different pairs of guide stars. A lot of effort has gone into computing those offsets so that all the images taken at a particular pointing can be combined successfully with astrometry matched to external astrometric catalogs. Unfortunately, there is no current mechanism for passing those updated solutions along to the community without providing entirely new copies of all the data.

4.3.2 Source Image

Any science observation with a valid WCS described by the FITS standard or any of the WCS conventions implemented in pywcs may serve as a source for creating a headerlet.

We describe as an example the type of WCS information of a typical HST ACS/WFC image as it is distributed by the HST archive (OTFR) after being processed with the latest image calibrations, including applying the latest available distortion models. The full description of the WCS is available in the "Distortion Correction In HST Files" report [Hack] by Hack et al. The science header now contains the following set of keywords and extensions to fully describe the WCS with distortion:

- **Linear WCS keywords**: CRPIX, CRVAL, CTYPE, CD matrix keywords
- **SIP coefficients**: A_*_* and B_*_*, A_ORDER, B_ORDER
- **The first order coefficients from the IDC table**: (needed by astrodrizzle) OCX10, OCX11, OCY10, and OCY11 keywords
- **NPOL distortion: if an NPOLFILE has been specified for the image,** CPDIS and DP record-value keywords to point to WCSDVARR extensions (Distortion Paper [Calabretta])
- **Detector defect correction: (for ACS/WFC this is a column defect)if a D2IMFILE has been** specified for use with the image, the D2IMEXT, D2IMERR and AXISCORR keywords point to the D2IMARR extension

Some of the distortion information may be stored in additional extensions in the science file:

- **WCSDVARR extensions: 2 extensions for each chip with lookup tables containing** the non-polynomial corrections, with each extension corresponding to an axis of the image (X correction or Y correction)
- **D2IMARR extension**: an extension with a lookup table containing the row- or column-correction from the D2IMFILE.

Each science header will have its own set of these keywords and extensions that will be kept together as part of the headerlet definition. This avoids any ambiguity as to what solution was used for any given WCS.

An HST ACS/WFC exposure would end up with the following set of extensions:

```
EXT#  FITSNAME      FILENAME               EXTVE DIMENS       BITPI OBJECT

0     j8hw27c4q_flt j8hw27c4q_flt.fits                        16
1       IMAGE       SCI                    1     4096x2048    -32
2       IMAGE       ERR                    1     4096x2048    -32
3       IMAGE       DQ                     1     4096x2048    16
4       IMAGE       SCI                    2     4096x2048    -32
5       IMAGE       ERR                    2     4096x2048    -32
6       IMAGE       DQ                     2     4096x2048    16
7       IMAGE       D2IMARR                1     4096         -32
8       IMAGE       WCSDVARR               1     65x33        -32
9       IMAGE       WCSDVARR               2     65x33        -32
10      IMAGE       WCSDVARR               3     65x33        -32
11      IMAGE       WCSDVARR               4     65x33        -32
```

These additional extensions add approximately 100kB to a typical ACS/WFC image making them a space efficient means of managing all the distortion and WCS information.

4.3.3 Headerlet Definition

A `headerlet` is a self-consistent definition of a single WCS including all distortion for all chips/detectors of a single exposure. This is different from alternate WCS defined in Greisen, E. W., and Calabretta (Paper I) [Greisen] in that by definition all alternate WCSs share the same distortion model while headerlets may be based on different distortion models. A headerlet does not include alternate WCSs. It is stored as a multi-extension FITS file following the structure of the science file. The WCS information in the science header is saved in the header of an HDU with EXTNAME 'SIPWCS'. All other HDUs in the headerlet (containing distorion information) have the same EXTNAME as the science file.

SIPWCS - A New FITS Extension

We introduce a new HDU with EXTNAME `SIPWCS`. It has no data and the header contains all the WCS-related keywords from the SCI header. As a minimum it contains the basic WCS keywords described in Paper I [Greisen] If the science observation has a SIP distortion model, the SIP keywords are included in this extension. If the distortion includes a non-polynomial part, the keywords describing the extensions with the lookup tables (EXTNAME=WCSDVARR) are also in this header. If there's a detector defect correction (row or column correction), the keywords describing the D2IMARR HDU are also in this header. In addition, each SIPWCS header contains two keywords which point back to the HDU of the original science file which was the source for it. These keywords are TG_ENAME and TG_EVER and have the meaning of (extname, extver) for the data extension in the science file.

The keywords in this extension are used by the software to overwrite the keywords in the corresponding SCI header to update the WCS solution for each chip without any computation.

Headerlet File Structure

This SIPWCS extension along with all WCSDVARR extensions and the D2IMARR extension if available fully describe the WCS of each chip. The listing of the FITS extensions for a `headerlet` for a sample ACS/WFC exposure after writing it out to a file is:

```
EXT#  FITSNAME      FILENAME               EXTVE DIMENS       BITPI OBJECT

0     j8hw27c4q     j8hw27c4q_hdr.fits                      16
```

(continues on next page)

(continued from previous page)

```
1       IMAGE       SIPWCS              1                   8
2       IMAGE       SIPWCS              2                   8
3       IMAGE       WCSDVARR            1       65x33       -32
4       IMAGE       WCSDVARR            2       65x33       -32
5       IMAGE       WCSDVARR            3       65x33       -32
6       IMAGE       WCSDVARR            4       65x33       -32
7       IMAGE       D2IMARR             1       4096        -32
```

Note: A headerlet derived from a full-frame WFC3/UVIS image would only contain a PRIMARY header and two SIPWCS extensions (one for each SCI extension) as WFC3/UVIS does not currently have non-polynomial distortion or any detector defect corrections.

The keywords used to populate the headerlet come from all the extensions of the updated FITS file, as illustrated in the following figure.

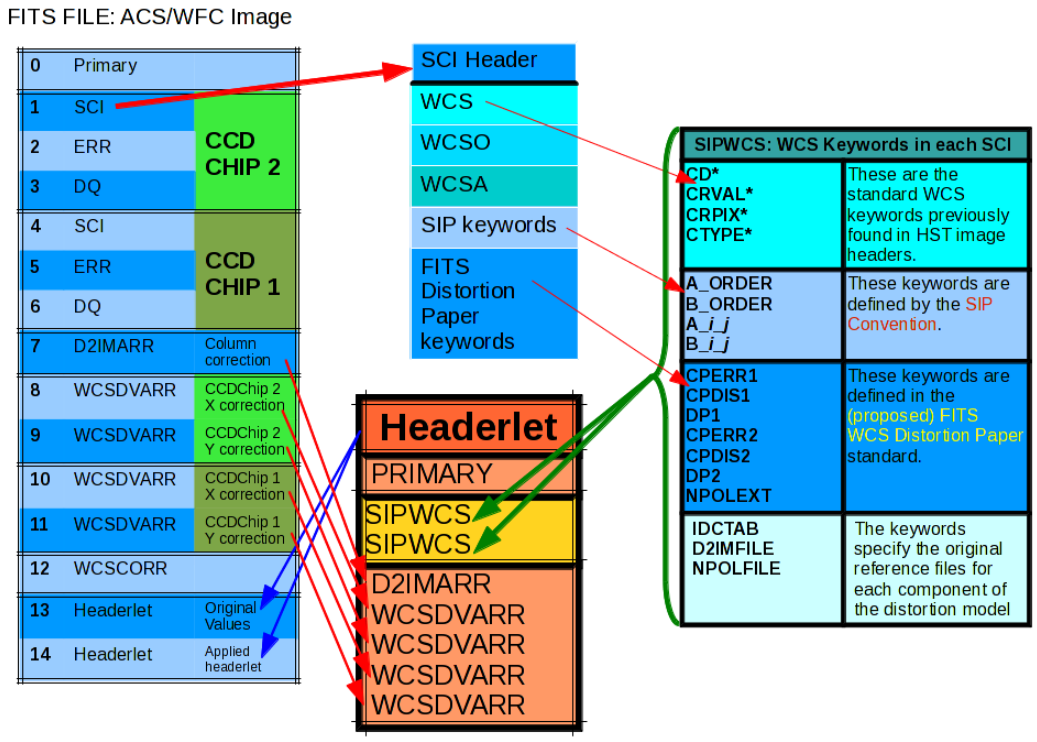


Fig. 4: This figure shows the keywords that are included in a headerlet, the extensions included in a headerlet, and how a headerlet appears as a new extension when it gets appended to the original ACS/WFC file.

Headerlet Primary Header

The list below contains all keywords specific to the primary header of a headerlet with a brief description how to determine their value. Note that all keywords will be present in the header and 'required' and 'optional' below refers to their value.

- **HDRNAME - (required) a unique name for the headerlet**
  - the value is given by the user as a parameter to *create_headerlet* or *write_headerlet*
  - HDRNAME<wcskey> from the science file is used
  - WCSNAME<wcskey> from the science file is used
  - KeyError is raised
- **DESTIM - (required) target image filename - used to determine if a headerlet can be applied to a science file.**
  - the ROOTNAME keyword of the original science file
  - the name of the science file
- **WCSNAME - (required) name for the WCS**
  - the value is given by the user as a parameter to *create_headerlet* or *write_headerlet*
  - WCSNAME<wcskey> from the science file is used
  - the value of hdrname parameter is used
  - HDRNAME<wcskey> from the science file
  - KeyError is raised
- **DISTNAME - (optional) name of distortion model**
  - **The value of DISTNAME has the form** <idctab rootname>-<npolfile rootname>-<d2imfile rootname> and has a value of 'NONE' if no reference files are specified.
- **SIPNAME - (optional) name of SIP model**
  - SIPNAME is constructed as <ROOTNAME>_<IDCTAB_rootname>, where

    ROOTNAME is the keyword from the science file header (or the file name)

    IDCTAB_rootname is the rootname of the idctab file

    so for example, SIPNAME for a science file j94f05bgq_flt.fits and an idctab file postsm4_idc.fits is j94f05bgq_postsm4
  - If the SIP coefficients are present in the header but IDCTAB is missing or invalid, then SIPNAME is set to UNKNOWN.
  - If there's no polynomial model, SIPNAME is set to NOMODEL.
- NPOLFILE - (optional) name of npol reference file
  - NPOLFILE keyword from science file primary header
  - UNKNOWN if NPOLFILE keyword is missing or invalid but data extensions exist
  - or NOMODEL
- IDCTAB - (optional)

  IDCTAB keyword from science file primary header or N/A
- D2IMFILE - (optional)

D2IMFILE keyword from science file primary header or N/A

- AUTHOR - (optional) name of person who created the headerlet
- DESCRIP - (optional) short description of the headerlet solution
- NMATCH - (optional) number of sources used in the new solution fit, if updated from the Archive's default WCS
- CATALOG - (optional) a reference frame used to define the astrometric solution
- UPWCSVER - (optional) version of STWCS used to create the WCS of the original image
- PYWCSVER - (optional) version of PyWCS used to create the WCS of the original image

These keywords are used to determine whether a headerlet can be applied to a given exposure or not. Some of the keywords provide more information about the solution itself, how it was derived, and by whom.

4.3.4 Working With Headerlets

Headerlets are implemented in a python module *headerlet* which uses PyWCS for WCS management and PyFITS for FITS file handling. The functionality includes methods to:

- **Create a headerlet (on disk or in memory) from a specific WCS of a science observation.** This can be the Primary or an alternate WCS.
- **Apply a WCS from a headerlet to the Primary WCS of a science observation (and** optionally save the original WCS as an alternate WCs or a different headerlet).
- Copy a WCS from a headerlet as an alternate WCS.
- Attach a headerlet to a science file.
- Archive a WCS of a science file as a headerlet attached to the file.
- Delete a headerlet attached to a science file.
- Print a summary of all headerlets attached to a science file.

An optional GUI interface is available through TEAL and includes functions for writing a headerlet, applying a headerlet, etc. A full listing of all functions with GUI interface is available after `stwcs.wcsutil` is imported.

The headerlet API as of the time of writing this report is documented in *Appendix 1: Headerlet API*.

`Note: For an up-to-date API always consult the current the SSB documentation pages.`

Headerlet HDU - A New Type of FITS Extension

The word `headerlet` has been used sofar in three different ways:

- A single WCS representation
- The multi-extension FITS file storing a WCS
- The extension of a science file containing a headerlet (as a WCS representation)

The last usage of the term `headerlet` is discussed in this section. When a `headerlet` is applied to an image, a copy of the original headerlet file is appended to the image's HDU list as a special extension HDU called a `HeaderletHDU`. A HeaderletHDU consists of a simple header describing the headerlet, and has as its data the headerlet file itself, (which may be compressed). A HeaderletHDU has an 'XTENSION' value of 'HDRLET'. Support for this is provided through the implementation of a NonstandardExtHDU in PyFITS.

When opening a file that contains `Headerlet HDU` extensions, it will normally look like this in PyFITS:

```
>>> import pyfits
>>> hdul = pyfits.open('94f05bgq_flt_with_hlet.fits')
>>> hdul.info()
Filename: j94f05bgq_flt_with_hlet.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     248   ()            int16
1    SCI         ImageHDU       286   (4096, 2048)  float32
2    ERR         ImageHDU        76   (4096, 2048)  float32
3    DQ          ImageHDU        66   (4096, 2048)  int16
4    SCI         ImageHDU       282   (4096, 2048)  float32
5    ERR         ImageHDU        74   (4096, 2048)  float32
6    DQ          ImageHDU        66   (4096, 2048)  int16
7    WCSCORR     BinTableHDU     56   10R x 23C     [40A, I, 1A, D, D, D, D, D, D, D,␣
↪D, 24A, 24A, D, D, D, D, D, D, D, J, 40A]
8    WCSDVARR    ImageHDU        15   (65, 33)      float32
9    WCSDVARR    ImageHDU        15   (65, 33)      float32
10   WCSDVARR    ImageHDU        15   (65, 33)      float32
11   WCSDVARR    ImageHDU        15   (65, 33)      float32
12   D2IMARR     ImageHDU        12   (4096,)       float32
13   HDRLET   NonstandardExtHDU   13
14   HDRLET   NonstandardExtHDU   13
```

The names of the `headerlet` extensions are both HDRLET, but its type shows up as `NonstandardExtHDU`. Their headers can be read, and while their data can be read you'd have to know what to do with it (the data is actually either a tar file or a gzipped tar file containing the `headerlet` file). However, if you have *`stwcs.wcsutil.headerlet`* imported, PyFITS will recognize these extensions as `Headerlet HDUs`:

```
>>> import stwcs.wcsutil.headerlet
>>> # Note that it's necessary to reopen the file
>>> hdul = pyfits.open('j94f05bgq_flt_with_hlet.fits')
>>> hdul.info()
Filename: j94f05bgq_flt_with_hlet.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     248   ()            int16
1    SCI         ImageHDU       286   (4096, 2048)  float32
2    ERR         ImageHDU        76   (4096, 2048)  float32
3    DQ          ImageHDU        66   (4096, 2048)  int16
4    SCI         ImageHDU       282   (4096, 2048)  float32
5    ERR         ImageHDU        74   (4096, 2048)  float32
6    DQ          ImageHDU        66   (4096, 2048)  int16
7    WCSCORR     BinTableHDU     56   10R x 23C     [40A, I, 1A, D, D, D, D, D, D, D,␣
↪D, 24A, 24A, D, D, D, D, D, D, D, J, 40A]
8    WCSDVARR    ImageHDU        15   (65, 33)      float32
9    WCSDVARR    ImageHDU        15   (65, 33)      float32
10   WCSDVARR    ImageHDU        15   (65, 33)      float32
11   WCSDVARR    ImageHDU        15   (65, 33)      float32
12   D2IMARR     ImageHDU        12   (4096,)       float32
13   HDRLET      HeaderletHDU    13
14   HDRLET      HeaderletHDU    13
>>> print hdul['HDRLET', 1].header.ascard
XTENSION= 'HDRLET  '           / Headerlet extension
BITPIX  =                    8 / array data type
NAXIS   =                    1 / number of array dimensions
NAXIS1  =               102400 / Axis length
PCOUNT  =                    0 / number of parameters
GCOUNT  =                    1 / number of groups
EXTNAME = 'HDRLET  '           / name of the headerlet extension
```

(continues on next page)

(continued from previous page)

```
HDRNAME = 'j94f05bgq_orig'      / Headerlet name
DATE    = '2011-04-13T12:14:42' / Date FITS file was generated
SIPNAME = 'IDC_qbu1641sj'       / SIP distortion model name
NPOLFILE= '/grp/hst/acs/lucas/new-npl/qbu16424j_npl.fits' / Non-polynomial correction
D2IMFILE= '/grp/hst/acs/lucas/new-npl/wfc_ref68col_d2i.fits' / Column correction
COMPRESS=                    F / Uses gzip compression
```

`HeaderletHDU` objects are similar to other HDU objects in PyFITS. However, they have a special `.headerlet` attribute that returns the actual `headerlet` contained in the HDU data as a `Headerlet` object:

```
>>> hdrlet = hdul['HDERLET', 1].headerlet
>>> hdrlet.info()
Filename: (No file associated with this HDUList)
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      12   ()            uint8
1    SIPWCS      ImageHDU       111   ()            uint8
2    SIPWCS      ImageHDU       110   ()            uint8
3    WCSDVARR    ImageHDU        15   (65, 33)      float32
4    WCSDVARR    ImageHDU        15   (65, 33)      float32
5    WCSDVARR    ImageHDU        15   (65, 33)      float32
6    WCSDVARR    ImageHDU        15   (65, 33)      float32
7    D2IMARR     ImageHDU        12   (4096,)       float32
```

This is useful if you want to view the contents of the `headerlets` attached to a file.

Examples

To create a `headerlet` from an image, a `createHeaderlet()` function is provided:

```
>>> from stwcs.wcsutil import headerlet
>>> hdrlet = headerlet.createHeaderlet('j94f05bgq_flt.fits', 'VERSION1')
>>> type(hdrlet)
<class 'stwcs.wcsutil.headerlet.Headerlet'>
>>> hdrlet.info()
Filename: (No file associated with this HDUList)
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      12   ()
1    SIPWCS      ImageHDU       111   ()
2    SIPWCS      ImageHDU       110   ()
3    WCSDVARR    ImageHDU        15   (65, 33)      float32
4    WCSDVARR    ImageHDU        15   (65, 33)      float32
5    WCSDVARR    ImageHDU        15   (65, 33)      float32
6    WCSDVARR    ImageHDU        15   (65, 33)      float32
7    D2IMARR     ImageHDU        12   (4096,)       float32
```

As you can see, the `Headerlet` object is similar to a normal PyFITS `HDUList` object. `createHeaderlet()` can be given either the path to a file, or an already open `HDUList` as its first argument.

What do you do with a `Headerlet` object? Its main purpose is to apply its WCS solution to another file. This can be done using the `Headerlet.apply()` method:

```
>>> hdrlet.apply('some_other_image.fits')
```

Or you can use the `applyHeaderlet()` convenience function. It takes an existing `headerlet` file path or object as its first argument; the rest of its arguments are the same as `Headerlet.apply()`. As with `createHeaderlet()` both of these can take a file path or opened `HDUList` objects as arguments.

When a `headerlet` is applied to an image, an additional `headerlet` containing that image's original WCS solution is automatically created, and is appended to the file's HDU list as a `Headerlet HDU`. However, this behavior can be disabled by setting the `createheaderlet` keyword argument to `False` in either `Headerlet.apply()` or `applyHeaderlet()`.

4.3.5 Appendix 1: Headerlet API

- *apply_headerlet_as_alternate*
- *apply_headerlet_as_primary*
- *archive_as_headerlet*
- *attach_headerlet*
- *create_headerlet*
- *delete_headerlet*
- *extract_headerlet*
- *print_summary*
- *restore_all_with_distname*
- *restore_from_headerlet*
- *write_headerlet*

apply_headerlet_as_alternate

```
def apply_headerlet_as_alternate(filename, hdrlet, attach=True, wcskey=None,
                                 wcsname=None, logging=False, logmode='w'):
    """
    Apply headerlet to a science observation as an alternate WCS

    Parameters
    ----------
    filename: string
             File name of science observation whose WCS solution will be updated
    hdrlet: string
             Headerlet file
    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'
    """
```

apply_headerlet_as_primary

```
def apply_headerlet_as_primary(filename, hdrlet, attach=True, archive=True,
                               force=False, logging=False, logmode='a'):
    """
    Apply headerlet 'hdrfile' to a science observation 'destfile' as the primary WCS

    Parameters
    ----------
    filename: string
             File name of science observation whose WCS solution will be updated
    hdrlet: string
             Headerlet file
    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
    """
```

archive_as_headerlet

```
def archive_as_headerlet(filename, hdrname, sciext='SCI',
                         wcsname=None, wcskey=None, destim=None,
                         sipname=None, npolfile=None, d2imfile=None,
                         author=None, descrip=None, history=None,
                         nmatch=None, catalog=None,
                         logging=False, logmode='w'):
    """
    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
```

(continues on next page)

(continued from previous page)

```
    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
    """
```

attach_headerlet

```
def attach_headerlet(filename, hdrlet, logging=False, logmode='a'):
    """
    Attach Headerlet as an HeaderletHDU to a science file

    Parameters
```

(continues on next page)

(continued from previous page)

```
    ----------
    filename: string, HDUList
            science file to which the headerlet should be applied
    hdrlet: string or Headerlet object
            string representing a headerlet file
    logging: boolean
            enable file logging
    logmode: 'a' or 'w'
    """
```

create_headerlet

```
def create_headerlet(filename, sciext='SCI', hdrname=None, destim=None,
                     wcskey=" ", wcsname=None,
                     sipname=None, npolfile=None, d2imfile=None,
                     author=None, descrip=None, history=None,
                     nmatch=None, catalog=None,
                     logging=False, logmode='w'):
    """
    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
```

(continues on next page)

(continued from previous page)

```
            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.
    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
    """
```

delete_headerlet

```
def delete_headerlet(filename, hdrname=None, hdrext=None, distname=None,
                     logging=False, logmode='w'):
    """
    Deletes HeaderletHDU(s) from a science file

    Notes
```

(continues on next page)

(continued from previous page)

```
    -----
    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

    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 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'
    """
```

extract_headerlet

```
def extract_headerlet(filename, output, extnum=None, hdrname=None,
                      clobber=False, logging=True):
    """
    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
```

(continues on next page)

(continued from previous page)

```
    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

    """
```

print_summary

```
def 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

    """
```

restore_all_with_distname

```
def restore_all_with_distname(filename, distname, primary, archive=True,
                              sciext='SCI', logging=False, logmode='w'):
    """
    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'
    """
```

restore_from_headerlet

```
def restore_from_headerlet(filename, hdrname=None, hdrext=None, archive=True,
                           force=False, logging=False, logmode='w'):
    """
    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'
    """
```

write_headerlet

```
def write_headerlet(filename, hdrname, output=None, sciext='SCI',
                        wcsname=None, wcskey=None, destim=None,
                        sipname=None, npolfile=None, d2imfile=None,
                        author=None, descrip=None, history=None,
                        nmatch=None, catalog=None,
                        attach=True, clobber=False, logging=False):

    """
    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.
```

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
        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 5

Indices and tables

- genindex
- modindex
- search

Bibliography

[DistortionPaper] Calabretta M. R., Valdes F. G., Greisen E. W., and Allen S. L., 2004, "Representations of distortions in FITS world coordinate systems",[cited 2012 Sept 18], Available from: http://www.atnf.csiro.au/people/mcalabre/WCS/dcs_20040422.pdf

[SIPConvention] Shupe D.L., Hook R.N., 2008, "The SIP Convention for Representing Distortion in FITS Image Headers", [cited 2012 Sept 18], Available from: http://fits.gsfc.nasa.gov/registry/sip.html

[Anderson2002] Anderson, J. 2002, in the Proceedings of the 2002 HST Calibration Workshop, S. Arribas, A. Koekemoer, and B. Whitmore, eds

[Calabretta2004] (draft FITS WCS Distortion paper) Calabretta M. R., Valdes F. G., Greisen E. W., and Allen S. L., 2004, "Representations of distortions in FITS world coordinate systems",[cited 2012 Sept 18], Available from: http://www.atnf.csiro.au/people/mcalabre/WCS/dcs_20040422.pdf

[Hack] Hack, et al, STScI 2012-01 TSR, http://stsdas.stsci.edu/tsr

[Calabretta] (draft FITS WCS Distortion paper) Calabretta M. R., Valdes F. G., Greisen E. W., and Allen S. L., 2004, "Representations of distortions in FITS world coordinate systems",[cited 2012 Sept 18], Available from: http://www.atnf.csiro.au/people/mcalabre/WCS/dcs_20040422.pdf

[Greisen] Greisen, E. W., and Calabretta M.R. 2002, A&A, 395 (Paper I)

Python Module Index

u

stwcs.updatewcs.astrometry_utils, 18
stwcs.updatewcs.corrections, 16
stwcs.updatewcs.det2im, 16

w

stwcs.wcsutil.altwcs, 9
stwcs.wcsutil.headerlet, 22
stwcs.wcsutil.hstwcs, 3

Index

Symbols

__version__ (*in module stwcs*), 20

A

all_world2pix() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 3
apply_as_alternate() (*stwcs.wcsutil.headerlet.Headerlet method*), 23
apply_as_primary() (*stwcs.wcsutil.headerlet.Headerlet method*), 23
apply_astrometric_updates() (*in module stwcs.updatewcs.astrometry_utils*), 19
apply_headerlet_as_alternate() (*in module stwcs.wcsutil.headerlet*), 26
apply_headerlet_as_primary() (*in module stwcs.wcsutil.headerlet*), 26
apply_new_apriori() (*stwcs.updatewcs.astrometry_utils.AstrometryDB method*), 18
archive_as_headerlet() (*in module stwcs.wcsutil.headerlet*), 26
archiveWCS() (*in module stwcs.wcsutil.altwcs*), 9
AstrometryDB (*class in stwcs.updatewcs.astrometry_utils*), 18
attach_headerlet() (*in module stwcs.wcsutil.headerlet*), 27
attach_to_file() (*stwcs.wcsutil.headerlet.Headerlet method*), 23
available_wcskeys() (*in module stwcs.wcsutil.altwcs*), 10

B

build_d2imname() (*in module stwcs.updatewcs.utils*), 17
build_distname() (*in module stwcs.updatewcs.utils*), 17
build_distname() (*stwcs.wcsutil.headerlet.Headerlet method*), 23
build_npolname() (*in module stwcs.updatewcs.utils*), 17
build_sipname() (*in module stwcs.updatewcs.utils*), 17

C

CompSIP (*class in stwcs.updatewcs.corrections*), 16
create_headerlet() (*in module stwcs.wcsutil.headerlet*), 28

D

delete_headerlet() (*in module stwcs.wcsutil.headerlet*), 29
deleteWCS() (*in module stwcs.wcsutil.altwcs*), 10
DET2IMCorr (*class in stwcs.updatewcs.det2im*), 16

E

equal_distmodel() (*stwcs.wcsutil.headerlet.Headerlet method*), 24
extract_headerlet() (*in module stwcs.wcsutil.headerlet*), 29

F

find_headerlet_HDUs() (*in module stwcs.wcsutil.headerlet*), 30
findObservation() (*stwcs.updatewcs.astrometry_utils.AstrometryDB method*), 19
format() (*stwcs.wcsutil.headerlet.FuncNameLoggingFormatter method*), 22
fromfile() (*stwcs.wcsutil.headerlet.Headerlet class method*), 24
fromheaderlet() (*stwcs.wcsutil.headerlet.HeaderletHDU class method*), 25
fromstring() (*stwcs.wcsutil.headerlet.Headerlet class method*), 24
FuncNameLoggingFormatter (*class in stwcs.wcsutil.headerlet*), 22

G

get_destination_model() (*stwcs.wcsutil.headerlet.Headerlet method*), 24
get_extname_extver_list() (*in module stwcs.wcsutil.headerlet*), 30
get_header_kw_vals() (*in module stwcs.wcsutil.headerlet*), 30
get_headerlet_kw_names() (*in module stwcs.wcsutil.headerlet*), 30
getKeyFromName() (*in module stwcs.wcsutil.altwcs*), 11
getObservation() (*stwcs.updatewcs.astrometry_utils.AstrometryDB method*), 19

H

Headerlet (*class in stwcs.wcsutil.headerlet*), 23
headerlet (*stwcs.wcsutil.headerlet.HeaderletHDU attribute*), 25
headerlet_summary() (*in module stwcs.wcsutil.headerlet*), 31
HeaderletHDU (*class in stwcs.wcsutil.headerlet*), 25
HSTWCS (*class in stwcs.wcsutil.hstwcs*), 3
hverify() (*stwcs.wcsutil.headerlet.Headerlet method*), 24

I

info() (*stwcs.wcsutil.headerlet.Headerlet method*), 24
init_attrs() (*stwcs.wcsutil.headerlet.Headerlet method*), 24
init_logging() (*in module stwcs.wcsutil.headerlet*), 31
is_par_blank() (*in module stwcs.wcsutil.headerlet*), 31
isAvailable() (*stwcs.updatewcs.astrometry_utils.AstrometryDB method*), 19

N

naxis1 (*stwcs.wcsutil.hstwcs.HSTWCS attribute*), 7
naxis2 (*stwcs.wcsutil.hstwcs.HSTWCS attribute*), 7
next_wcskey() (*in module stwcs.wcsutil.altwcs*), 11
NPOLCorr (*class in stwcs.updatewcs.npol*), 16

P

parse_filename() (*in module stwcs.wcsutil.headerlet*), 31
pc2cd() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
print_summary() (*in module stwcs.wcsutil.headerlet*), 32
printwcs() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7

R

readIDCCoeffs() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
readModel() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
readModelFromIDCTAB() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
resetLTV() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
restore_all_with_distname() (*in module stwcs.wcsutil.headerlet*), 32
restore_from_headerlet() (*in module stwcs.wcsutil.headerlet*), 32
restoreWCS() (*in module stwcs.wcsutil.altwcs*), 10

S

setInstrSpecKw() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 7
setOrient() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 8
setPscale() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 8
stwcs.updatewcs.astrometry_utils (*module*), 18
stwcs.updatewcs.corrections (*module*), 16
stwcs.updatewcs.det2im (*module*), 16
stwcs.wcsutil.altwcs (*module*), 9
stwcs.wcsutil.headerlet (*module*), 22
stwcs.wcsutil.hstwcs (*module*), 3
summary() (*stwcs.wcsutil.headerlet.Headerlet method*), 24

T

TDDCorr (*class in stwcs.updatewcs.corrections*), 15
tofile() (*stwcs.wcsutil.headerlet.Headerlet method*), 25

U

update_ref_files() (*in module stwcs.wcsutil.headerlet*), 32
update_versions() (*in module stwcs.wcsutil.headerlet*), 33
updateObs() (*stwcs.updatewcs.astrometry_utils.AstrometryDB method*), 19
updatePscale() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 8
updatewcs() (*in module stwcs.updatewcs*), 14

V

VACorr (*class in stwcs.updatewcs.corrections*), 15
verify_dest() (*stwcs.wcsutil.headerlet.Headerlet method*), 25
verify_hdrname() (*stwcs.wcsutil.headerlet.Headerlet method*), 25
verify_hdrname_is_unique() (*in module stwcs.wcsutil.headerlet*), 33

W

wcs2header() (*stwcs.wcsutil.hstwcs.HSTWCS method*), 8
wcskeys() (*in module stwcs.wcsutil.altwcs*), 10
wcsnames() (*in module stwcs.wcsutil.altwcs*), 10
with_logging() (*in module stwcs.wcsutil.headerlet*), 33
write_headerlet() (*in module stwcs.wcsutil.headerlet*), 33