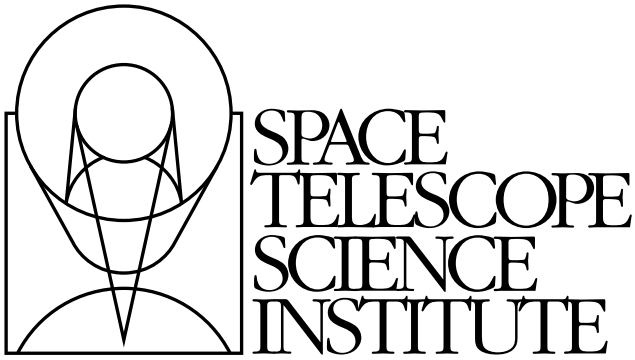


DrizzlePac Documentation

Release 2.1.3.dev50623 (1-Feb-2016)

Warren Hack, Nadia Dencheva, Chris Sontag,
Megan Sosey, Michael Droettboom, Mihai Cara

April 18, 2016

This package supports the use of *AstroDrizzle* as an integrated set of modules that can be run in an automated manner to combine images, along with other tasks to support image alignment and coordinate transformations with distortion included. The version of *DrizzlePac* described here implements a single task to run the entire *AstroDrizzle* processing pipeline, while also providing the framework for users to create their own custom pipeline based on the modules in this package merged with their own custom code if desired. These pages document what functions and classes are available for use under Python while providing the syntax for calling those functions from Python tasks.

Full documentation of how to run the primary *AstroDrizzle* and *TweakReg* tasks, along with fully worked examples, can be found in the DrizzlePac Handbook.

This package relies on the STWCS package in order to provide the support for the WCS-based distortion models and alignment of the input images.

Contents:

CHAPTER

ONE

PRIMARY USER INTERFACE: ASTRODRIZZLE()

`AstroDrizzle` - Python implementation of *MultiDrizzle*

`AstroDrizzle` automates the process of aligning images in an output frame, identifying cosmic-rays, removing distortion, and then combining the images after removing the identified cosmic-rays.

This process involves a number of steps, namely:

1. Processing the input images and input parameters
2. Creating a static mask
3. Performing sky subtraction
4. Drizzling onto separate output images
5. Creating the median image
6. Blotting the median image
7. Identifying and flagging cosmic-rays
8. Final combination

A full description of this process can be found in the DrizzlePac Handbook.

Output
The primary output from this task is the distortion-corrected, cosmic-ray cleaned, and combined image as a FITS file.

This task requires numerous user-settable parameters to control the primary aspects of each of the processing steps.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

`drizzlepac.astrodrizzle.`**`AstroDrizzle`**(*input=None*, *mdriztab=False*, *editpars=False*, *configobj=None*, *wcsmap=None*, ***input_dict*)

Parameters

input : str or list of str (Default = `'*flt.fits'`)

The name or names of the input files to be processed, which can be provided in any of the following forms:

- filename of a single image
- filename of an association (`ASN`) table
- wild-card specification for files in directory
- comma-separated list of filenames
- `@file` filelist containing list of desired input filenames. The file list needs to be provided as an ASCII text file containing a list of filenames for all input images with one filename on each line of the file. If inverse variance maps (IVM maps) have also been created by the user and are to be used (by specifying 'IVM' to the parameter final_wht_type), then these are simply provided as a second column in the filelist, with each IVM filename listed on the same line as a second entry, after its corresponding exposure filename.

Note: If the user specifies `IVM` for the *final_wht_type*, but does not provide the names of `IVM` files, *AstroDrizzle* will automatically generate the IVM files itself for each input exposure.

Output : str (Default = '')

The rootname for the output drizzled products. This step can result in the creation of several files, including:

- copies of each input image as a `FITS` image, if *workinplace*'='*Yes* and/or input images are in `GEIS` format.
- mask files and coeffs files created by *PyDrizzle* for use by *drizzle*.

If an association file has been given as input, the specified filename will be used instead of the product name specified in the ASN file. Similarly, if a single exposure is provided, the rootname of the single exposure will be used for the output product instead of relying on the input rootname. If no value is provided when a filelist or wild-card specification is given as input, then a rootname of 'final' will be used for the output file name.

mdriztab : bool (Default = False)

This button will immediately update the parameter values in the TEAL GUI based on those provided by the `MDRIZTAB` reference table referenced in the first input image. This requires that the `MDRIZTAB` reference file be available locally.

runfile : str (Default = 'astrodrizzle.log')

This log file will contain all the output messages generated during processing, including full details of any errors/exceptions. These messages will be a super-set of those reported to the screen during processing.

wcskey : str (Default = '')

This parameter corresponds to the *key* for the WCS being selected by the user. It allows the user to select which WCS solution should be used for processing the images when multiple WCS's have been updated in each input image header using the Paper I Multiple WCS FITS standard.

> **Warning:** Use of this parameter should be done only when all input images have been updated using the Paper I FITS standard for specifying Multiple WCS's in each image header. This parameter assumes that the same WCS letter corresponds to WCS's that have been updated in a consistent manner. For example, all input images have been updated to be consistent with their distortion model in the WCS's with key of *A*.

proc_unit : str (Default = 'native')

The units to be used for the final output drizzled product. Valid values and definitions are:

- **native**: Output DRZ product and input 'values' given in the native units of the input image.
- **electrons**: Output DRZ product and input 'values' given in units of electrons.

coeffs : bool (Default = Yes)

This parameter determines whether or not to use the coefficients stored in the each input image header. If turned off, no distortion coefficients will be applied during the coordinate transformations.

context : bool (Default = Yes)

This parameter specifies whether or not to create a context image during the final drizzle combination. The context image contains the information regarding which image(s) contributed to each pixel encoded as a bit-mask. More information on context images can be obtained from the ACS Data Handbook.

group : int (Default = None)

This parameter establishes whether or not a single FITS extension, or group will be drizzled. If an extension is provided, then only that chip will be drizzled onto the output frame. Either a FITS extension number, a GEIS group number (such as '1'), or a FITS extension name (such as 'sci,1') may be specified.

build : bool (Default = No)

When this parameter is set to 'Yes' (True), AstroDrizzle will combine the separate 'drizzle' output files into a single multi-extension format FITS file. This

combined output file will contain seperate SCI (science), WHT (weight), and CTX (context) extensions. If this parameter is set to 'No' (False), a separate simple FITS file will be created for each aforementioned extension.

crbit : int (Default = 4096)

This parameter sets the bit value for CR identification in the DQ array.

stepsize : int (Default = 10)

This parameter controls the internal grid of points used in the coordinate transformation from the input image to the output frame. The default value of 10 indicates that every 10th pixel will be transformed using the full WCS-based transformation. All remaining pixels will then be transformed using bilinear interpolation based on those pixels (i.e. every 10th pixel in the case of the default parameter setting) that were fully transformed.

resetbits : int (Default = 4096)

This parameter allows the user to specify which DQ bits of each input image DQ array should be reset to a value of 0. This operation is performed on the copy of the input data after updating the headers based on the 'updatewcs' parameter, and prior to starting any of the AstroDrizzle processing steps (static mask, sky subtraction, and so on).

num_cores: int (Default = None)

This specifies the number of CPU cores to use during processing. Any value less than 2 will disable all use of parallel processing.

in_memory: bool (Default = False)

This parameter sets whether or not to keep all intermediate products in memory when processing. This includes all single drizzle products (*single_sci and *single_wht), median image, blot images, and crmask images. The use of this option will therefore require significantly more memory than usual to process the data while reducing the overall processing time by eliminating most of the disk activity. *Only* the products of the final drizzle step will get written out when this parameter gets specified as 'True'.

STATE OF INPUT FILES

restore: bool (Default = No)

Setting this to 'Yes' (True) directs AstroDrizzle to copy the input images from the 'OrIg_files' sub-directory and use them for processing, if they had been archived by AstroDrizzle using the 'preserve' or 'overwrite' parameters already. If set to 'Yes' and the input files had not been archived already, it will simply ignore this and work with the current input images.

preserve : bool (Default = Yes)

Copy input files to archive directory, if not already archived. This parameter determines whether or not astrodrizzle creates a copy of the input file in a

sub-directory called 'OrIg_files'. If a copy already exists in this directory, then the previously existing version will NOT be overwritten.

overwrite : bool (Default = No)

Copy input files into archive, overwriting older files if required? This parameter will cause astrodrizzle to make a copy of each input file in the 'OrIg_files' directory regardless of whether a previous copy existed or not, and will overwrite any previous copy should it be present.

clean : bool (Default = No)

The temporary files created by AstroDrizzle can be automatically removed by setting this parameter to 'Yes' (True). The affected files include the coefficient and static mask files created by PyDrizzle, in addition to other intermediate files created by AstroDrizzle. It is often useful to retain the intermediate files and examine them when first learning how to run AstroDrizzle. However, when running AstroDrizzle routinely, or on a small disk drive, these files can be removed to conserve space.

STEP 1: STATIC MASK

static : bool (Default = Yes)

Create a static bad-pixel mask from the data? This mask flags all pixels that deviate by more than a value of 'static_sig' sigma below the image median, since these pixels are typically the result of bad pixel oversubtraction in the dark image during calibration.

static_sig : float (Default = 4.0)

The number of sigma below the RMS to use as the clipping limit for creating the static mask.

STEP 2: SKY SUBTRACTION

skysub : bool (Default = Yes)

Turn on or off sky subtraction on the input data. When *skysub* is set to *no*, then *skyuser* field will be enabled and if user specifies a header keyword showing the sky value in the image, then that value will be used for CR-rejection but it will not be subtracted from the (drizzled) image data. If user sets *skysub* to *yes* then *skyuser* field will be disabled (and if it is not empty - it will be ignored) and user can use one of the methods available through the *skymethod* parameter to compute the sky or provide a file (see *skyfile* parameter) with values that should be subtracted from (single) drizzled images.

skymethod : {'localmin', 'globalmin+match', 'globalmin', 'match'} (Default = 'localmin')

Select the algorithm for sky computation:

- **'localmin'**: compute a common sky for all members of *an exposure*. For a typical use, it will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in an input image,

and it will subtract the previously found minimum sky value from all chips (marked for sky subtraction) in that image. This process is repeated for each input image.

Note: This setting is recommended when regions of overlap between images are dominated by "pure" sky (as opposite to extended, diffuse sources).

Note: This is similar to the "skysub" algorithm used in previous versions of astrodrizzle.

- **'globalmin'**: compute a common sky value for all members of **all** "sky-lines". It will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in **all** input images, find the minimum sky value, and then it will subtract the **same** minimum sky value from **all** chips (marked for sky subtraction) in **all** images. This method *may* useful when input images already have matched background values.
- **'match'**: compute differences in sky values between images in common (pair-wise) sky regions. In this case computed sky values will be relative (delta) to the sky computed in one of the input images whose sky value will be set to (reported to be) 0. This setting will "equalize" sky values between the images in large mosaics. However, this method is not recommended when used in conjunction with astrodrizzle because it computes relative sky values while *astrodrizzle* needs "measured" sky values for median image generation and CR rejection.
- **'globalmin+match'**: first find a minimum "global" sky value in all input images and then use **'match'** method to equalize sky values between images.

Note: This is the *recommended* setting for images containing diffuse sources (e.g., galaxies, nebulae) covering significant parts of the image.

---

skywidth : float (Default = 0.3)

Bin width, in sigma, used to sample the distribution of pixel flux values in order to compute the sky background statistics.

skystat : {'median', 'mode', 'mean'} (Default = 'median')

Statistical method for determining the sky value from the image pixel values.

skylower : float (Default = None)

Lower limit of usable pixel values for computing the sky. This value should be specified in the units of the input image(s).

skyupper : float (Default = None)

Upper limit of usable pixel values for computing the sky. This value should be specified in the units of the input image(s).

skyclip : int (Default = 5)

Number of clipping iterations to use when computing the sky value.

skylsigma : float (Default = 4.0)

Lower clipping limit, in sigma, used when computing the sky value.

skyusigma : float (Default = 4.0)

Upper clipping limit, in sigma, used when computing the sky value.

skymask_cat : str (Default = '')

File name of a catalog file listing user masks to be used with images.

use_static : bool (Default = True)

Specifies whether or not to use static mask to exclude masked image pixels from sky computations.

sky_bits : int, str, None (Default = 0)

Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for sky computations. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for sky computations, then *sky_bits* should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting *sky_bits* to 12.

Default value (0) will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for sky computations.

Set *sky_bits* to *None* to turn off the use of image's DQ array for sky computations.

In order to reverse the meaning of the *sky_bits* parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for sky computations and to consider as "good" all other pixels

(regardless of their DQ flag), set *sky_bits* to `~4+8`, or `~4,8`. To obtain the same effect with an *int* input value (except for 0), enter -(4+8+1)=-9. Following this convention, a *sky_bits* string value of `'~0'` would be equivalent to setting `sky_bits=None`.

Note: DQ masks (if used), *will be* combined with user masks specified in the input @-file.

Note: To summarize, below are provided allowable syntaxes for *sky_bits*:

- Specify that bits 4,8, and 512 be considered "good" bits and that all other bits be considered "bad" bits:
  - Integer: 524 [numerically: 4+8+512=524]
  - String: 4+8+512
  - String: 4,8,512
- Specify that only bits 4,8, and 512 be considered "bad" bits and that all other bits be considered "good" bits:
  - Integer: -525 [numerically: ~(4+8+512)=~524=-(524+1)=-525]
  - String: ~4+8+512 or ~(4+8+512)
  - String: ~4,8,512 or ~(4,8,512)

---

skyfile : str (Default = '')

Name of file containing user-computed sky values to be used with each input image. This ASCII file should only contain 2 columns: image filename in column 1 and sky value in column 2 (and higher hor multi-chip images). The sky value should be provided in units that match the units of the input image and for multi-chip images, if only one sky value was provided in column 2, the same value will be applied to all chips. If more than one sky value are provided (in columns 2, 3, ...) then the number of sky values should match the number of SCI extensions in the images.

skyuser : str (Default = '')

Name of header keyword which records the sky value already subtracted from the image by the user. The *skyuser* parameter is ignored when *skysub* is set to *yes*.

Alternatively, user can enter the name of a file that contains user-computed sky values. To distinguish a file name from a header keyword, prepend '@' to the file name. For example '@my_sky_values.txt'. The format of the file with user-supplied sky values is the same as that of a *skyfile*.

Note: When *skysub*'='*no* and *skyuser* field is empty, then `AstroDrizzle` will assume that sky background is 0.0 for the purpose of cosmic-ray rejection.

STEP 3: DRIZZLE SEPARATE IMAGES

driz_separate : bool (Default = Yes)

This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the same WCS as the final combined output frame. These images are used to create the median image, needed for cosmic ray rejection.

driz_sep_kernel : str {'square','point','gaussian','turbo','tophat','lanczos3'} (Default = 'turbo')

Used for the initial separate drizzling operation only, this parameter specifies the form of the kernel function used to distribute flux onto the separate output images. The current options are:

- **square**: original classic drizzling kernel
- **point**: this kernel is a point so each input pixel can only contribute to the single pixel that is closest to the output position. It is equivalent to the limit as pixfrac -> 0, and is very fast.
- **gaussian**: this kernel is a circular gaussian with a FWHM equal to the value of pixfrac, measured in input pixels.
- **turbo**: this is similar to kernel="square" but the box is always the same shape and size on the output grid, and is always aligned with the X and Y axes. This may result in a significant speed increase.
- **tophat**: this kernel is a circular "top hat" shape of width pixfrac. It effects only output pixels within a radius of pixfrac/2 from the output position.
- **lanczos3**: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form of the "sinc" interpolator, and is very effective for resampling single images when scale=pixfrac=1. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

Warning: The ''lanczos3" kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for pixfrac != 1.0, and is not recommended for scale != 1.0.

The default for this step is **"turbo"** since it is much faster than **"square"**, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

driz_sep_wt_scl : float (Default = exptime)

This parameter specifies the weighting factor for input image. If driz_sep_wt_scl=exptime, then the scaling value will be set equal to the exposure time found in the image header. The use of the Default Value is recommended for producing optimal behavior for most scenarious. It is possible to set wt_scl=expsq for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

driz_sep_pixfrac : float (Default = 1.0)

Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If pixfrac is set to less than 0.001, the kernel parameter will be reset to 'point' for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the 'drizzle' task.

driz_sep_fillval : int (Default = None)

Value to be assigned to output pixels that have zero weight, or that receive flux from any input pixels during drizzling. This parameter corresponds to the 'fillval' parameter of the 'drizzle' task. If the default of 'None' is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that value.

driz_sep_bits : int, str, or None (Default = 0)

Integer sum of all the DQ bit values from the input image's DQ array that should be considered 'good' when building the weighting mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of AstroDrizzle, by adding the value 4096 for ACS and WFPC2 data. For possible input formats, see the description for *sky_bits* parameter.

STEP 3a: CUSTOM WCS FOR SEPARATE OUTPUTS

driz_sep_wcs : bool (Default = No)

Define custom WCS for seperate output images?

driz_sep_refimage : str (Default = '')

Reference image from which a WCS solution can be obtained.

driz_sep_rot : float (Default = None)

Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of None specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled

image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

driz_sep_scale : float (Default = None)

Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of None specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

driz_sep_outnx : int (Default = None)

Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_outny : int (Default = None)

Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_ra : float (Default = None)

Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_dec : float (Default = None)

Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

STEP 4: CREATE MEDIAN IMAGE

median : bool (Default = Yes)

This parameter specifies whether or not to create a median image. This median image will be used as the comparison 'truth' image :in the cosmic ray rejection step.

median_newmasks : bool (Default = Yes)

This parameter specifies whether or not new mask files will be created when the median image is created. These masks are generated from weight files previously produced by the "driz_separate" step, and contain all bad pixel information used to exclude pixels when calculating the median. Generally this step should be set to 'Yes' (True), unless for some reason, it is desirable to include bad pixel information when generating the median.

combine_maskpt : float (Default = 0.3)

Percentage of weight image values, below which the are flagged.

combine_type : str {'median', 'mean', 'minmed', 'imedian', 'imean', 'iminmed'} (Default = 'minmed')

This parameter defines the method that will be used to create the median image. The 'mean' and 'median' options set the calculation type when running 'numcombine', a numpy method for median-combining arrays to create the median image. The "minmed" option will produce an image that is generally the same as the median, except in cases where the median is significantly higher than the minimum good pixel value. In this case, "minmed" will choose the minimum value. The sigma thresholds for this decision are provided by the "combine_nsigma" parameter. However, as the "combine_nsigma" parameter does not adjust for the larger probability of a single "nsigma" event with a greater number of images, "minmed" will bias the comparison image low for a large number of images. "minmed" is highly recommended for three images, and is good for four to six images, but should be avoided for ten or more images.

A value of 'median' is the recommended method for a large number of images, and works equally well as minmed down to approximately four images. However, the user should set the "combine_nhigh" parameter to a value of 1 when using "median" with four images, and consider raising this parameter's value for larger numbers of images. As a median averages the two inner values when the number of values being considered is even, the user may want to keep the total number of images minus "combine_nhigh" odd when using "median".

The options starting with 'i', such as 'imedian', works just like the normal median operation except when dealing with a pixel were all the values are flagged as 'bad'. In this case, the 'i' functions return the last pixel in the stack as if it were good. This will prevent saturated pixels in the image from leaving holes in the middle of the stars, for example.

combine_nsigma : float (Default = '4 3')

This parameter defines the sigmas used for accepting minimum values, rather than median values, when using the 'minmed' combination method. If two values are specified the first value will be used in the initial choice between median and minimum, while the second value will be used in the "growing" step to reject additional pixels around those identified in the first step. If only one value is specified, then it is used in both steps.

combine_nlow : int (Default = 0)

This parameter sets the number of low value pixels to reject automatically during image combination.

combine_nhigh : int (Default = 0)

This parameter sets the number of high value pixels to reject automatically during image combination.

combine_lthresh : float (Default = None)

Sets the lower threshold for clipping input pixel values during image combination. This value gets passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_hthresh : float (Default = None)

This parameter sets the upper threshold for clipping input pixel values during image combination. The value for this parameter is passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_grow : int (Default = 1)

Width, in pixels, beyond the limit set by the rejection algorithm being used, for additional pixels to be rejected in an image. This parameter is used to set the 'grow' parameter in 'imcombine' for use in creating the median image.

combine_bufsize : float (Default = None)

Size of buffer, in Mb, to use when reading in each section of each input image. The default buffer size is 1Mb. The larger the buffer size, the fewer times the code needs to open each input image and the more memory will be required to create the median image. A larger buffer can be helpful when using compression, since slower copies need to be made of each set of rows from each input image instead of using memory-mapping.

STEP 5: BLOT BACK THE MEDIAN IMAGE

blot : bool (Default = Yes)

Perform the blot operation on the median image? If set to 'Yes' (True), the output will be median smoothed images that match each input chips location, and will be used in the cosmic ray rejection step.

blot_interp : str{'nearest', 'linear', 'poly3', 'poly5', 'sinc'} (Default = 'poly5')

This parameter defines the method of interpolation to be used when blotting drizzled images back to their original WCS solution. Valid options include:

- **nearest**: Nearest neighbor
- **linear**: Bilinear interpolation in x and y
- **poly3**: Third order interior polynomial in x and y
- **poly5**: Fifth order interior polynomial in x and y
- **sinc**: Sinc interpolation (accurate but slow)

The 'poly5' interpolation method has been chosen as the default because it is relatively fast and accurate.

If 'sinc' interpolation is selected, then the value of the parameter for 'blot_sinscl' will be used to specify the size of the sinc interpolation kernel.

blot_sinscl : float (Default = 1.0)

Size of the sinc interpolation kernel in pixels.

blot_addsky : bool (Default = Yes)

Add back a sky value using the MDRIZSKY value from the header. If 'Yes' (True), the blot_skyval parameter is ignored.

blot_skyval : float (Default = 0.0)

This is a user-specified custom sky value to be added to the blot image. This is only used if blot_addsky is 'No' (False).

STEP 6: REMOVE COSMIC RAYS WITH DERIV, DRIZ_CR

driz_cr : bool (Default = Yes)

Perform cosmic-ray detection? If set to 'Yes' (True), cosmic-rays will be detected and used to create cosmic-ray masks based on the algorithms from 'deriv' and 'driz_cr'.

driz_cr_corr : bool (Default = No)

Create a cosmic-ray cleaned input image? If set to 'Yes' (True), a cosmic-ray cleaned _crclean image will be generated directly from the input image, and a corresponding _crmask file will be written to document detected pixels affected by cosmic-rays.

driz_cr_snr : list of floats (Default = '3.5 3.0')

The values for this parameter specify the signal-to-noise ratios for the 'driz_cr' task to be used in detecting cosmic rays. See the help file for 'driz_cr' for further discussion of this parameter.

driz_cr_grow : int (Default = 1)

The radius, in pixels, around each detected cosmic-ray, in which more stringent detection criteria for additional cosmic rays will be used.

driz_cr_ctegrow : int (Default = 0)

Length, in pixels, of the CTE tail that should be masked in the drizzled output.

driz_cr_scale : str (Default = '1.2 0.7')

Scaling factor applied to the derivative in 'driz_cr' when detecting cosmic-rays. See the help file for 'driz_cr' for further discussion of this parameter.

STEP 7: DRIZZLE FINAL COMBINED IMAGE

driz_combine : bool (Default = Yes)

This parameter specifies whether or not to drizzle each input image onto the final output image. This applies the generated cosmic-ray masks to the input images and creates a final, cleaned, distortion-corrected image.

final_wht_type : {'EXP', 'ERR', 'IVM'} (Default = 'EXP')

Specify the type of weighting image to apply with the bad pixel mask

for the final drizzle step. The options for this parameter include:

- **EXP**: The default of 'EXP' indicates that the images will be weighted according to their exposure time, which is the standard behavior for drizzle. This weighting is a good approximation in the regime where the noise is dominated by photon counts from the sources, while contributions from sky background, read-noise and dark current are negligible. This option is provided as the default since it produces reliable weighting for all types of data, including older instruments (eg., WFPC2), where more sophisticated options may not be available.

- **ERR**: Specifying 'ERR' is an alternative for ACS and STIS data. In these cases, the final drizzled images will be weighted according to the inverse variance of each pixel in the input exposure files, calculated from the error array data extension that is in each calibrated input exposure file. This array is exposure time dependent, and encapsulates all of the noise sources in each exposure including read-noise, dark current, sky background, and Poisson noise from the sources themselves. For WFPC2, the ERR array is not produced during the calibration process, and therefore is not a viable option. We advise extreme caution when selecting the "ERR" option, since the nature of this weighting scheme can introduce photometric discrepancies in sharp unresolved sources, although these effects are minimized for sources with gradual variations between pixels. The "EXP" weighting option does not suffer from these effects, and is therefore the recommended option.

- **IVM**: Specifying 'IVM' allows the user to either supply their own inverse-variance weighting map, or allow AstroDrizzle to generate one automatically on-the-fly during the final drizzle step. This parameter option may be necessary for specific purposes. For example, to create a drizzled weight file for software such as SExtractor, it is expected that a weight image containing all of the background noise sources (sky level, read-noise, dark current, etc), but not the Poisson noise from the objects themselves will be available. The user can create the inverse variance images and then specify their names using the 'input' parameter for AstroDrizzle to specify an '@file'. This would be a single ASCII file containing the list of input calibrated exposure filenames (one per line), with a second column containing the name of the IVM file corresponding to each calibrated exposure. Each IVM file must have the same file format as the input file, and if provided as multi-extension FITS files (e.g., ACS or STIS data) then the IVM extension must have the EXTNAME of 'IVM'. If no IVM files are specified on input, then AstroDrizzle will rely on the flat-field reference file and computed dark value from the image header to automatically generate an IVM file specific to each exposure.

final_kernel : {'square','point','gaussian','turbo','tophat','lanczos3'} (Default = 'square')

This parameter specifies the form of the kernel function used to distribute flux onto the separate output images, for the initial separate drizzling operation only. The value options for this parameter include:

- **square**: original classic drizzling kernel

- **point**: this kernel is a point so each input pixel can only contribute to

the single pixel that is closest to the output position. It is equivalent to the limit as pixfrac -> 0, and is very fast.

- **gaussian**: this kernel is a circular gaussian, measured in input pixels, with a FWHM value equal to the value of pixfrac.
- **turbo**: this is similar to kernel="square", except that the box is always the same shape and size on the output grid, and is always aligned with the X and Y axes. This may result in a significant speed increase.
- **tophat**: this kernel is a circular "top hat" shape of width pixfrac. It effects only output pixels within a radius of pixfrac/2 from the output position.
- **lanczos3**: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form of the "sinc" interpolator, and is very effective for resampling single images when scale=pixfrac=1. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

> **Warning:** The ''lanczos3" kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for pixfrac != 1.0, and is not recommended for scale != 1.0.

The default for this step is **"turbo"** since it is much faster than **"square"**, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

final_wt_scl : float (Default = exptime)

This parameter specifies the weighting factor for input image. If final_wt_scl=exptime, then the scaling value will be set equal to the exposure time found in the image header. The use of the Default Value is recommended for producing optimal behavior for most scenarious. It is possible to set wt_scl=expsq for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

final_pixfrac : float (Default = 1.0)

Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If pixfrac is set to less than 0.001, the kernel parameter will be reset to 'point' for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the 'drizzle' task.

final_fillval : float (Default = None)

The value for this parameter is to be assigned to the output pixels that have zero weight or which do not receive flux from any input pixels during drizzling. This parameter corresponds to the 'fillval' parameter of the 'drizzle' task. If the default of 'None' is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that numerical value.

final_bits : int, str, or None (Default = 0)

Integer sum for all of the DQ bit values from the input image's DQ array that should be considered 'good' when building the weight mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of AstroDrizzle, by adding the value 4096 for ACS and WFPC2 data. For possible input formats, see the description for *sky_bits* parameter.

final_units : str (Default = 'cps')

This parameter determines the units of the final drizzle-combined image, and can either be 'counts' or 'cps'. It is passed through to 'drizzle' in the final drizzle step.

STEP 7a: CUSTOM WCS FOR FINAL OUTPUT

final_wcs : bool (Default = No)

Obtain the WCS solution from a user-designated reference image?

final_refimage : str (Default = '')

Reference image from which a WCS solution can be obtained. If no extension is specified (such as 'sci,1' or '4'), then astrodrizzle will automatically look for the FIRST extension which contains a valid HSTWCS object to read in as the WCS. Otherwise, the user can explicitly provide the extension name for multi-extension FITS files.

final_rot : float (Default = None)

Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of None specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

final_scale : float (Default = None)

Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of None specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

final_outnx : int (Default = None)

Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

final_outny : int (Default = None)

Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

final_ra : float (Default Valut = None)

Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

final_dec : float (Default Valut = None)

Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

INSTRUMENT PARAMETERS

gain : float (Default = None)

Value used to override instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the ''gainkeyword" parameter is in use.

gainkeyword : str (Default = '')

Keyword used to specify a value, which is used to override the instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the ''gain" parameter is in use.

rdnoise : float (Default = None)

Value used to override instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the ''rnkeyword" parameter is in use.

rnkeyword : str (Default = '')

Keyword used to specify a value, which is used to override the instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the ''rdnoise" parameter is in use.

exptime : float (Default = None)

Value used to override default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the ''expkeyword" parameter is in use.

expkeyword : str (Default = '')

Keyword used to specify a value, which is used to override the default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the ''exptime" parameter is in use.

ADVANCED PARAMETERS AVAILABLE FROM COMMAND LINE

updatewcs : bool (Default = No)

This parameter specifies whether the WCS keywords are to be updated by running updatewcs on the input data, or left alone. The update performed by updatewcs not only recomputes the WCS based on the currently used `IDCTAB`, but also populates the header with the `SIP` coefficients. For `ACS/WFC` images, the time-dependence correction will also be applied to the `WCS` and `SIP` keywords. This parameter should be set to 'No' (*False*) when the WCS keywords have been carefully set by some other method, and need to be passed through to drizzle 'as is', otherwise those updates will be over-written by this update.

Note: This parameter was preserved in the API for compatibility purposes with existing user processing pipe-lines. However, it has been removed from the TEAL interface because it is easy to have it set to 'yes' (especially between consecutive runs of *AstroDrizzle*) with potentially disastrous effects on input image WCS (for example it could wipe-out previously aligned WCS).

See also:

drizzlepac.adrizzle
Apply the 'drizzle' algorithm to the images

drizzlepac.ablot
Apply the 'blot' algorithm to drizzled images

drizzlepac.sky
Perform sky subtraction

skypac.skymatch
Sky computation and equalization

drizzlepac.median
Create a median combined image from a set of drizzled images

drizzlepac.drizCR
Identify cosmic-rays by comparing blotted, median images to the original input images

Notes

Something to keep in mind is that the full AstroDrizzle interface will make backup copies of your original files and place them in the `OrIg_files/` directory of you current working directory.

All calibrated input images must have been updated using *updatewcs* from the *STWCS* package, to include the full distortion model in the header. Alternatively, one can set *updatewcs* parameter to *True*

when running either *TweakReg* or `AstroDrizzle` from command line (Python interpreter) **the first time** on such images.

Examples

The astrodrizzle task can be run from either the TEAL GUI or from the command-line using PyRAF or Python. These examples illustrate the various syntax options available.

Example 1: Drizzle a set of calibrated (`_flt.fits`) images using mostly default parameters. Select the 'World Coordinate System' keyword to the updated solution computed from *tweakreg*. When combining images, ignore pixel flags of 64 and 32 in the DQ array of the (`_flt.fits`) images. Align the final product such that North is up, and set the final pixel scale to 0.05 arcseconds/pixel.

1. Run the task from PyRAF using the TEAL GUI:

```
>>> import drizzlepac
>>> epar astrodrizzle
```

2. Run the task from PyRAF using the command line.

```
>>> import drizzlepac
>>> from drizzlepac import astrodrizzle
>>> astrodrizzle.AstroDrizzle('*flt.fits', output='final',
...     wcskey='TWEAK', driz_sep_bits='64,32', final_wcs=True,
...     final_scale=0.05, final_rot=0)
```

Or, run the same task from the PyRAF command line, but specify all parameters in a config file named `myparam.cfg`:

```
>>> astrodrizzle.AstroDrizzle('*flt.fits', configobj='myparam.cfg')
```

3. Run the task directly from Python:

```
>>> from drizzlepac import astrodrizzle
>>> astrodrizzle.AstroDrizzle('*flt.fits', output='final',
...     wcskey='TWEAK', driz_sep_bits='64,32', final_wcs=True,
...     final_scale=0.05, final_rot=0)
```

4. Help can be accessed via the "Help" pulldown menu in the TEAL GUI. It can also be accessed from the PyRAF command-line and saved to a text file:

```
>>> from drizzlepac import astrodrizzle
>>> astrodrizzle.help()
```

or

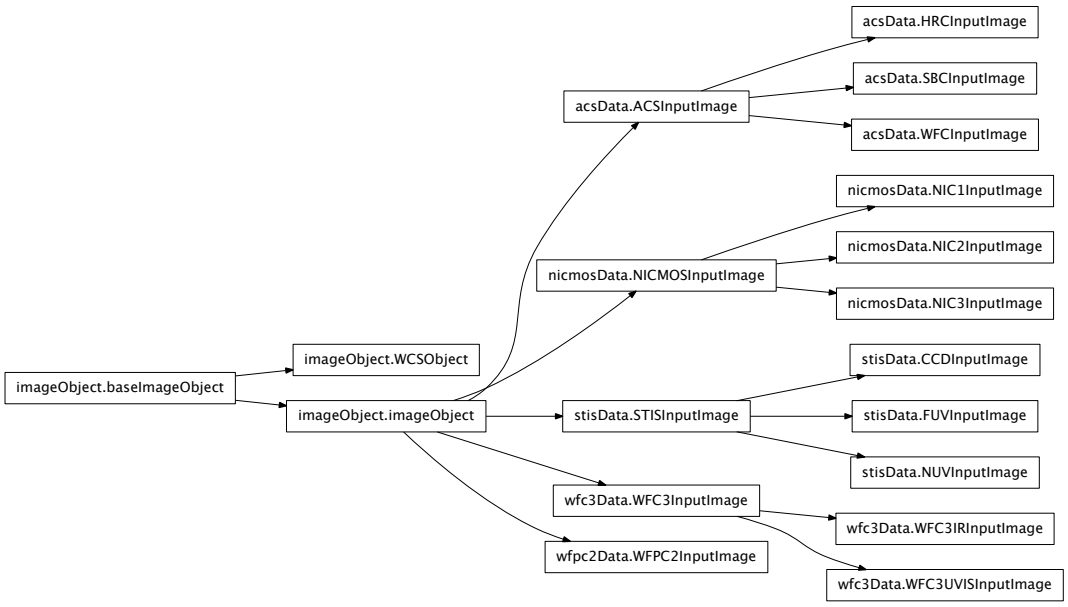
```
>>> astrodrizzle.help(file='help.txt')
>>> page help.txt
```

CHAPTER

TWO

IMAGEOBJECT CLASSES

This class and related sub-classes manage all the instrument-specific images for processing by *AstroDrizzle*.



2.1 Base ImageObject Classes

A class which makes image objects for each input filename. A class which makes image objects for each input filename.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.imageObject.**baseImageObject**(*filename*)

Bases: object

Base ImageObject which defines the primary set of methods.

buildERRmask(*chip*, *dqarr*, *scale*)

Builds a weight mask from an input DQ array and an ERR array associated with the input image.

buildEXPmask(*chip*, *dqarr*)

Builds a weight mask from an input DQ array and the exposure time per pixel for this chip.

buildIVMmask(*chip*, *dqarr*, *scale*)

Builds a weight mask from an input DQ array and either an IVM array provided by the user or a self-generated IVM array derived from the flat-field reference file associated with the input image.

buildMask(*chip*, *bits=0*, *write=False*)

Build masks as specified in the user parameters found in the configObj object.

We should overload this function in the instrument specific implementations so that we can add other stuff to the badpixel mask? Like vignetting areas and chip boundries in nicmos which are camera dependent? these are not defined in the DQ masks, but should be masked out to get the best results in multidrizzle.

clean()

Deletes intermediate products generated for this imageObject.

close()

Close the object nicely and release all the data arrays from memory YOU CANT GET IT BACK, the pointers and data are gone so use the getData method to get the data array returned for future use. You can use putData to reattach a new data array to the imageObject.

findExtNum(*extname=None*, *extver=1*)

Find the extension number of the give extname and extver.

find_DQ_extension()

Return the suffix for the data quality extension and the name of the file which that DQ extension should be read from.

getAllData(*extname=None*, *exclude=None*)

This function is meant to make it easier to attach ALL the data extensions of the image object so that we can write out copies of the original image nicer.

If no extname is given, the it retrieves all data from the original file and attaches it. Otherwise, give the name of the extensions you want and all of those will be restored.

Ok, I added another option. If you want to get all the data extensions EXCEPT a particular one, leave extname=NONE and set exclude=EXTNAME. This is helpfull cause you might not know all the extnames the image has, this will find out and exclude the one you do not want overwritten.

getData(*exten=None*)

Return just the data array from the specified extension fileutil is used instead of fits to account for non- FITS input images. openImage returns a fits object.

getExtensions(*extname='SCI'*, *section=None*)
Return the list of EXTVER values for extensions with name specified in extname.

getGain(*exten*)

getHeader(*exten=None*)
Return just the specified header extension fileutil is used instead of fits to account for non-FITS input images. openImage returns a fits object.

getInstrParameter(*value*, *header*, *keyword*)
This method gets a instrument parameter from a pair of task parameters: a value, and a header keyword.

The default behavior is:

- if the value and header keyword are given, raise an exception.
- if the value is given, use it.
- if the value is blank and the header keyword is given, use the header keyword.
- if both are blank, or if the header keyword is not found, return None.

getKeywordList(*kw*)
Return lists of all attribute values for all active chips in the imageObject.

getNumpyType(*irafType*)
Return the corresponding numpy data type.

getOutputName(*name*)
Return the name of the file or PyFITS object associated with that name, depending on the setting of self.inmemory.

getReadNoiseImage(*chip*)

Notes

Method for returning the readnoise image of a detector (in electrons).

The method will return an array of the same shape as the image.

Units
electrons

getdarkcurrent()

Notes

Return the dark current for the detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Units
electrons

getdarkimg(*chip*)

Notes

Return an array representing the dark image for the detector.

The method will return an array of the same shape as the image.

Units
electrons

getexptimeimg(*chip*)

Returns
exptimeimg : numpy array

The method will return an array of the same shape as the image.

Notes

Return an array representing the exposure time per pixel for the detector. This method will be overloaded for IR detectors which have their own EXP arrays, namely, WFC3/IR and NICMOS images.

Units
None

getflat(*chip*)
Method for retrieving a detector's flat field.

Returns
flat: array

This method will return an array the same shape as the image in **units of electrons**.

getskyimg(*chip*)

Notes

Return an array representing the sky image for the detector. The value of the sky is what would actually be subtracted from the exposure by the skysub step.

Units
electrons

info()
Return fits information on the _image.

putData(*data=None*, *exten=None*)
Now that we are removing the data from the object to save memory, we need something that cleanly puts the data array back into the object so that we can write out everything together using something like fits.writeto....this method is an attempt to make sure that when you add an

array back to the .data section of the hdu it still matches the header information for that section (ie. update the bitpix to reflect the datatype of the array you are adding). The other header stuff is up to you to verify.

Data should be the data array exten is where you want to stick it, either extension number or a string like 'sci,1'

returnAllChips (*extname=None*, *exclude=None*)
Returns a list containing all the chips which match the extname given minus those specified for exclusion (if any).

saveVirtualOutputs (*outdict*)
Assign in-memory versions of generated products for this imageObject based on dictionary 'outdict'.

set_mt_wcs (*image*)
Reset the WCS for this image based on the WCS information from another imageObject.

set_units ()
Record the units for this image, both BUNITS from header and in_units as needed internally. This method will be defined specifically for each instrument.

set_wtscl (*chip*, *wtscl_par*)
Sets the value of the wt_scl parameter as needed for drizzling.

updateContextImage (*contextpar*)
Reset the name of the context image to None if parameter *context* == False.

updateData (*exten*, *data*)
Write out updated data and header to the original input file for this object.

updateIVMName (*ivmname*)
Update outputNames for image with user-supplied IVM filename.

updateOutputValues (*output_wcs*)
Copy info from output WCSObject into outputnames for each chip for use in creating outputimage object.

class drizzlepac.imageObject.**imageObject** (*filename*, *group=None*, *inmemory=False*)
Bases: *drizzlepac.imageObject.baseImageObject*

This returns an imageObject that contains all the necessary information to run the image file through any multidrizzle function. It is essentially a PyFits object with extra attributes.

There will be generic keywords which are good for the entire image file, and some that might pertain only to the specific chip.

compute_wcslin (*undistort=True*)
Compute the undistorted WCS based solely on the known distortion model information associated with the WCS.

setInstrumentParameters (*instrpars*)
Define instrument-specific parameters for use in the code. By definition, this definition will need to be overridden by methods defined in each instrument's sub-class.

set_units(*chip*)
Define units for this image.

class drizzlepac.imageObject.**WCSObject**(*filename*, *suffix='_drz'*)
Bases: *drizzlepac.imageObject.baseImageObject*

restore_wcs()

2.2 ACS ImageObjects

Class used to model ACS specific instrument data.

Authors
Christopher Hanley, Warren Hack, Ivo Busko, David Grumm

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.acsData.**ACSInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.imageObject.imageObject*

doUnitConversions()

getdarkcurrent(*extver*)
Return the dark current for the ACS detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Returns
darkcurrent: float

Dark current value for the ACS detector in **units of electrons**.

SEPARATOR = '_'

class drizzlepac.acsData.**WFCInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.acsData.ACSInputImage*

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

This method gets called from processInput.

class drizzlepac.acsData.**HRCInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.acsData.ACSInputImage*

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

This method gets called from processInput.

class drizzlepac.acsData.**SBCInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.acsData.ACSInputImage*

setInstrumentParameters(*instrpars*)
Sets the instrument parameters.

2.3 WFC3 ImageObjects

wfc3Data module provides classes used to import WFC3 specific instrument data.

Authors
Megan Sosey, Christopher Hanley

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.wfc3Data.**WFC3InputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.imageObject.imageObject*

SEPARATOR = '_'

class drizzlepac.wfc3Data.**WFC3UVISInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.wfc3Data.WFC3InputImage*

doUnitConversions()

getdarkcurrent(*chip*)
Return the dark current for the WFC3 UVIS detector. This value will be contained within an instrument specific keyword.

Returns
darkcurrent: float

The dark current value with **units of electrons**.

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.wfc3Data.**WFC3IRInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.wfc3Data.WFC3InputImage*

doUnitConversions()
WF3 IR data come out in electrons, and I imagine the photometry keywords will be calculated as such, so no image manipulation needs be done between native and electrons

getdarkcurrent()
Return the dark current for the WFC3/IR detector. This value will be contained within an instrument specific keyword.

Returns
darkcurrent: float

The dark current value in **units of electrons**.

getdarkimg(*chip*)
Return an array representing the dark image for the detector.

Returns
dark: array

Dark image array in the same shape as the input image with **units of cps**

getexptimeimg(*chip*)
Return an array representing the exposure time per pixel for the detector.

Returns
dark: array

Exposure time array in the same shape as the input image

getskyimg(*chip*)

Notes

Return an array representing the sky image for the detector. The value of the sky is what would actually be subtracted from the exposure by the skysub step.

Units
electrons

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

2.4 STIS ImageObjects

stisData module provides classes used to import STIS specific instrument data.

Authors
Megan Sosey, Christopher Hanley

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.stisData.**STISInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.imageObject.imageObject*

doUnitConversions()
Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getflat(*chip*)
Method for retrieving a detector's flat field. For STIS there are three. This method will return an array the same shape as the image.

SEPARATOR = '_'

class drizzlepac.stisData.**CCDInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.stisData.STISInputImage*

getReadNoise()
Method for returning the readnoise of a detector (in DN).

Units
DN

This should work on a chip, since different chips to be consistant with other detector classes where different chips have different gains.

getdarkcurrent()
Returns the dark current for the STIS CCD chip.

Returns
darkcurrent : float

Dark current value in **units of electrons** (or counts, if proc_unit=='native').

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.stisData.**NUVInputImage**(*filename*, *group=None*)
Bases: *drizzlepac.stisData.STISInputImage*

doUnitConversions()
Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()
Returns the dark current for the STIS NUV detector.

Returns
darkcurrent : float

Dark current value in **units of electrons** (or counts, if proc_unit=='native').

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.stisData.**FUVInputImage**(*filename=None*, *group=None*)
Bases: *drizzlepac.stisData.STISInputImage*

doUnitConversions()
Convert the data to electrons.

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()
Returns the dark current for the STIS FUV detector.

Returns
darkcurrent : float

Dark current value in **units of electrons** (or counts, if proc_unit=='native').

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

2.5 NICMOS ImageObjects

Class used to model NICMOS specific instrument data.

Authors
Christopher Hanley, David Grumm, Megan Sosey

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.nicmosData.**NICMOSInputImage**(*filename=None*)
Bases: *drizzlepac.imageObject.imageObject*

doUnitConversions()
Convert the data to electrons

This converts all science data extensions and saves the results back to disk. We need to make sure the data inside the chips already in memory is altered as well.

getdarkcurrent()
Return the dark current for the NICMOS detectors.

Returns
darkcurrent : float

Dark current value with **units of cps**.

getdarkimg(*chip*)
Return an array representing the dark image for the detector.

Returns
dark : array

The dark array in the same shape as the image with **units of cps**.

getexptimeimg(*chip*)
Return an array representing the exposure time per pixel for the detector.

Returns
dark: array

Exposure time array in the same shape as the input image

getflat(*chip*)
Method for retrieving a detector's flat field.

Returns
flat : array

The flat field array in the same shape as the input image with **units of cps**.

isCountRate()
isCountRate: Method or IRInputObject used to indicate if the science data is in units of counts or count rate. This method assumes that the keyword 'BUNIT' is in the header of the input FITS file.

SEPARATOR = '_'

class drizzlepac.nicmosData.**NIC1InputImage**(*filename=None*)
Bases: *drizzlepac.nicmosData.NICMOSInputImage*

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.nicmosData.**NIC2InputImage**(*filename=None*)
Bases: *drizzlepac.nicmosData.NICMOSInputImage*

createHoleMask()
Add in a mask for the coronographic hole to the general static pixel mask.

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

class drizzlepac.nicmosData.**NIC3InputImage**(*filename=None*)
Bases: *drizzlepac.nicmosData.NICMOSInputImage*

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

2.6 WFPC2 ImageObjects

wfpc2Data module provides classes used to import WFPC2 specific instrument data.

Authors
Warren Hack, Ivo Busko, Christopher Hanley

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.wfpc2Data.**WFPC2InputImage**(*filename*, *group=None*)
Bases: *drizzlepac.imageObject.imageObject*

buildMask(*chip*, *bits=0*, *write=False*)
Build masks as specified in the user parameters found in the configObj object.

doUnitConversions()
Apply unit conversions to all the chips, ignoring the group parameter. This insures that all the chips get the same conversions when this gets done, even if only 1 chip was specified to be processed.

find_DQ_extension()
Return the suffix for the data quality extension and the name of the file which that DQ extension should be read from.

getEffGain()
Method used to return the effective gain of a instrument's detector.

Returns
gain : float

The effective gain.

getReadNoise(*exten*)
Method for returning the readnoise of a detector (in counts).

Returns
readnoise : float

The readnoise of the detector in **units of counts/electrons**.

getdarkcurrent(*exten*)
Return the dark current for the WFPC2 detector. This value will be contained within an instrument specific keyword. The value in the image header will be converted to units of electrons.

Returns
darkcurrent : float

Dark current for the WFPC3 detector in **units of counts/electrons**.

getflat(*chip*)
Method for retrieving a detector's flat field.

Returns
flat : array

The flat-field array in the same shape as the input image.

setInstrumentParameters(*instrpars*)
This method overrides the superclass to set default values into the parameter dictionary, in case empty entries are provided.

SEPARATOR = '_'

CHAPTER

THREE

STEP 1: PROCESS INPUT

This module supports the interpretation and initial verification of all the input files specified by the user. These functions:

- reads in parameter values from MDRIZTAB reference file and merges those merges those values in with the rest of the parameters from the GUI/configObj, if use of MDRIZTAB was specified
- insure that all input files are multi-extension FITS files and converts them if they are not
- updates all input WCS's to be consistent with IDCTAB, if specified
- generates the ImageObject instances for each input file
- resets the DQ bits if specified by the user
- adds info about any user-provided IVM files to the ImageObjects
- generates the output WCS based on user inputs

Process input to MultiDrizzle/PyDrizzle.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

The input can be one of:

- a python list of files
- a comma separated string of filenames (including wild card characters)
- an association table
- an @file (can have a second column with names of ivm files)

No mixture of instruments is allowed. No mixture of association tables, @files and regular fits files is allowed. Files can be in GEIS or MEF format (but not waiver fits).

Runs some sanity checks on the input files. If necessary converts files to MEF format (this should not be left to *makewcs* because *updatewcs* may be *False*). Runs makewcs. The function `process_input` returns an association table, ivmlist, output name

The common interface interpreter for MultiDrizzle tasks, 'processCommonInput()', not only runs 'process_input()' but 'createImageObject()' and 'defineOutput()' as well to fully setup all inputs for use with the rest of the MultiDrizzle steps either as stand-alone tasks or internally to MultiDrizzle itself.

drizzlepac.processInput.**addIVMInputs**(*imageObjectList*, *ivmlist*)
Add IVM filenames provided by user to outputNames dictionary for each input imageObject.

drizzlepac.processInput.**applyContextPar**(*imageObjectList*, *contextpar*)
Apply the value of the parameter *context* to the input list, setting the name of the output context image to None if *context* is False

drizzlepac.processInput.**buildASNList**(*rootnames*, *asnname*, *check_for_duplicates=True*)
Return the list of filenames for a given set of rootnames

drizzlepac.processInput.**buildEmptyDRZ**(*input*, *output*)
Create an empty DRZ file.

This module creates an empty DRZ file in a valid FITS format so that the HST pipeline can handle the Multidrizzle zero expossure time exception where all data has been excluded from processing.

Parameters
input : str

filename of the initial input to process_input

output : str

filename of the default empty _drz.fits file to be generated

drizzlepac.processInput.**buildFileList**(*input*, *output=None*, *ivmlist=None*, *wcskey=None*, *updatewcs=True*, ***workinplace*)
Builds a file list which has undergone various instrument-specific checks for input to MultiDrizzle, including splitting STIS associations.

drizzlepac.processInput.**buildFileListOrig**(*input*, *output=None*, *ivmlist=None*, *wcskey=None*, *updatewcs=True*, ***workinplace*)
Builds a file list which has undergone various instrument-specific checks for input to MultiDrizzle, including splitting STIS associations. Compared to buildFileList, this version returns the list of the original file names as specified by the user (e.g., before GEIS->MEF, or WAIVER FITS->MEF conversion).

drizzlepac.processInput.**changeSuffixinASN**(*asnfile*, *suffix*)
Create a copy of the original asn file and change the name of all members to include the suffix.

drizzlepac.processInput.**checkDGEOFile**(*filenames*)
Verify that input file has been updated with NPOLFILE

This function checks for the presence of 'NPOLFILE' kw in the primary header when 'DGEOFILE' kw is present and valid (i.e. 'DGEOFILE' is not blank or 'N/A'). It handles the case of science files downloaded from the archive before the new software was installed there. If 'DGEOFILE' is present and 'NPOLFILE' is missing, print a message and let the user choose whether to (q)uit and update the

headers or (c)ontinue and run astrodrizzle without the non-polynomial correction. 'NPOLFILE' will be populated in the pipeline before astrodrizzle is run.

In the case of WFPC2 the old style dgeo files are used to create detector to image correction at runtime.

Parameters

filenames : list of str

file names of all images to be checked

drizzlepac.processInput.**checkForDuplicateInputs**(*rootnames*)

Check input files specified in ASN table for duplicate versions with multiple valid suffixes (_flt and _flc, for example).

drizzlepac.processInput.**checkMultipleFiles**(*input*)

Evaluates the input to determine whether there is 1 or more than 1 valid input file.

drizzlepac.processInput.**createImageObjectList**(*files*, *instrpars*, *group=None*, *undistort=True*, *inmemory=False*)

Returns a list of imageObject instances, 1 for each input image in the list of input filenames.

drizzlepac.processInput.**getMdriztabPars**(*input*)

High-level function for getting the parameters from MDRIZTAB

Used primarily for TEAL interface.

drizzlepac.processInput.**manageInputCopies**(*filelist*, ***workinplace*)

Creates copies of all input images in a sub-directory.

The copies are made prior to any processing being done to the images at all, including updating the WCS keywords. If there are already copies present, they will NOT be overwritten, but instead will be used to over-write the current working copies.

drizzlepac.processInput.**processFilenames**(*input=None*, *output=None*, *infilesOnly=False*)

Process the input string which contains the input file information and return a filelist,output

drizzlepac.processInput.**process_input**(*input*, *output=None*, *ivmlist=None*, *updatewcs=True*, *prodonly=False*, *wcskey=None*, ***workinplace*)

Create the full input list of filenames after verifying and converting files as needed.

drizzlepac.processInput.**reportResourceUsage**(*imageObjectList*, *outwcs*, *num_cores*, *interactive=False*)

Provide some information to the user on the estimated resource usage (primarily memory) for this run.

drizzlepac.processInput.**resetDQBits**(*imageObjectList*, *cr_bits_value=4096*)

Reset the CR bit in each input image's DQ array

drizzlepac.processInput.**runmakewcs**(*input*)

Runs make wcs and recomputes the WCS keywords

Parameters

input : str or list of str

a list of files

Returns

output : list of str

returns a list of names of the modified files (For GEIS files returns the translated names.)

drizzlepac.processInput.**setCommonInput**(*configObj*, *createOutwcs=True*)

The common interface interpreter for MultiDrizzle tasks which not only runs 'process_input()' but 'createImageObject()' and 'defineOutput()' as well to fully setup all inputs for use with the rest of the MultiDrizzle steps either as stand-alone tasks or internally to MultiDrizzle itself.

Parameters

configObj : object

configObj instance or simple dictionary of input parameters

imageObjectList : list of imageObject objects

list of imageObject instances, 1 for each input exposure

outwcs : object

imageObject instance defining the final output frame

Notes

At a minimum, the configObj instance (dictionary) should contain:

configObj = {'input':None,'output':None }

If provided, the configObj should contain the values of all the multidrizzle parameters as set by the user with TEAL. If no configObj is given, it will retrieve the default values automatically. In either case, the values from the input_dict will be merged in with the configObj before being used by the rest of the code.

Examples

You can set *createOutwcs=False* for the cases where you only want the images processed and no output wcs information in necessary; as in:

>>>imageObjectList,outwcs = processInput.processCommonInput(configObj)

drizzlepac.processInput.**update_member_names**(*oldasndict*, *pydr_input*)

Update names in a member dictionary.

Given an association dictionary with rootnames and a list of full file names, it will update the names in the member dictionary to contain '_*' extension. For example a rootname of 'u9600201m' will be replaced by 'u9600201m_c0h' making sure that a MEf file is passed as an input and not the corresponding GEIS file.

drizzlepac.processInput.**userStop**(*message*)

3.1 ResetBits Update of Input

This module provides the capability to set a specific set of bit values in the input DQ arrays to zero. This allows a user to reset pixels previously erroneously flagged as cosmic-rays to good for reprocessing with improved alignment or detection parameters. resetbits - A module to set the value of specified array pixels to zero

This module allows a user to reset the pixel values of any integer array, such as the DQ array from an HST image, to zero.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

3.1.1 PARAMETERS

filename
[str] full filename with path

bits
[str] sum or list of integers corresponding to all the bits to be reset

3.1.2 Optional Parameters

extver
[int, optional] List of version numbers of the arrays to be corrected (default: None, will reset all matching arrays)

extname
[str, optional] EXTNAME of the arrays in the FITS files to be reset (default: 'dq')

3.1.3 NOTES

This module performs a simple bitwise-and on all the pixels in the specified array and the integer value provided as input using the operation (array & ~bits).

3.1.4 Usage

It can be called not only from within Python, but also from the host-level operating system command line using the syntax:

```
resetbits filename bits [extver [extname]]
```

3.1.5 EXAMPLES

1. The following command will reset the 4096 bits in all the DQ arrays of the file 'input_flt.fits':

```
resetbits input_flt.fits 4096
```

   or from the Python command line:

```
>>> import resetbits
>>> resetbits.reset_dq_bits("input_file_flt.fits", 4096)
```

2. To reset the 2,32,64 and 4096 (sum of 4194) bits in the second DQ array, specified as 'dq,2', in the file 'input_flt.fits':

```
resetbits input_flt.fits 4194 2
```

   or from the Python command line:

```
>>> import resetbits
>>> resetbits.reset_dq_bits("input_file_flt.fits", 2+32+64+4096, extver=2)
```

drizzlepac.resetbits.**getHelpAsString**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

drizzlepac.resetbits.**help**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.resetbits.**reset_dq_bits**(*input*, *bits*, *extver=None*, *extname='dq'*)
This function resets bits in the integer array(s) of a FITS file.

Parameters
filename : str

full filename with path

bits : str

sum or list of integers corresponding to all the bits to be reset

extver : int, optional

List of version numbers of the DQ arrays to be corrected [Default Value: None, will do all]

extname : str, optional

EXTNAME of the DQ arrays in the FITS file [Default Value: 'dq']

Notes

The default value of None for the 'extver' parameter specifies that all extensions with EXTNAME matching 'dq' (as specified by the 'extname' parameter) will have their bits reset.

Examples

1. The following command will reset the 4096 bits in all the DQ arrays of the file input_file_flt.fits:

```
reset_dq_bits("input_file_flt.fits", 4096)
```

2. To reset the 2,32,64 and 4096 bits in the second DQ array, specified as 'dq,2', in the file input_file_flt.fits:

```
reset_dq_bits("input_file_flt.fits", "2,32,64,4096", extver=2)
```

drizzlepac.resetbits.**run**(*configobj=None*)

Teal interface for running this code.

CHAPTER

FOUR

STEP 2: GENERATE A STATIC MASK

This step focuses entirely on creating a static mask for each detector used in the input images of all negative (presumably bad) pixel values. This module provides functions and classes that manage the creation of the global static masks.

For *staticMask*, the user interface function is *createMask*.

Authors

Ivo Busko, Christopher Hanley, Warren Hack, Megan Sosey

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.staticMask.**staticMask**(*configObj=None*)

This class manages the creation of the global static mask which masks pixels that are unwanted in the SCI array. A static mask object gets created for each global mask needed, one for each chip from each instrument/detector. Each static mask array has type Int16, and resides in memory.

Notes

Class that manages the creation of a global static mask which is used to mask pixels that are some sigma BELOW the mode computed for the image.

addMember(*imagePtr=None*)

Combines the input image with the static mask that has the same signature.

Parameters

imagePtr : object

An imageObject reference

Notes

The signature parameter consists of the tuple:

```
(instrument/detector, (nx,ny), chip_id)
```

The signature is defined in the image object for each chip

close()

Deletes all static mask objects.

deleteMask(*signature*)

Delete just the mask that matches the signature given.

getFilename(*signature*)
Returns the name of the output mask file that should reside on disk for the given signature.

getMaskArray(*signature*)
Returns the appropriate StaticMask array for the image.

getMaskname(*chipid*)
Construct an output filename for the given signature:

```
signature=[instr+detector,(nx,ny),detnum]
```

The signature is in the image object and the name of the static mask file is saved as sci_chip.outputNames["staticMask"].

saveToFile(*imageObjectList*)
Saves the static mask to a file it uses the signatures associated with each mask to contruct the filename for the output mask image.

drizzlepac.staticMask.**buildSignatureKey**(*signature*)
Build static file filename suffix used by mkstemp()

drizzlepac.staticMask.**constructFilename**(*signature*)
Construct an output filename for the given signature:

```
signature=[instr+detector,(nx,ny),detnum]
```

The signature is in the image object.

drizzlepac.staticMask.**createMask**(*input=None*, *static_sig=4.0*, *group=None*, *editpars=False*, *configObj=None*, ***inputDict*)
Create a static mask for all input images. The mask contains pixels that fall more than *static_sig* RMS below the mode for a given chip or extension. Those severely negative, or low pixels, might result from oversubtraction of bad pixels in the dark image, or high sky levels during calibration. For example, each ACS WFC image contains a separate image for each of 2 CCDs, and seperate masks will be generated for each chip accordingly.

The final static mask for each chip contains all of the bad pixels that meet this criteria from all of the input images along with any bad pixels that satisfy the final_bits value specified by the user, and found in the images DQ mask.

Users should consider the details of their science image and decide whether or not creating this mask is appropriate for their resulting science. For example, if your field is very crowded, or contains mostly nebulous or extended objects, then the statistcs could be heavily skewed and the mask could end up containing sources.

The generated static masks are saved to disk for use in later steps with the following naming convention:

[Instrument][Detector]_[xsize]x[ysize]_[detector number]_staticMask.fits

so an ACS image would produce a static mask with the name:

ACSWFC_2048x4096_1_staticMask.fits

and this would be the only file saved to disk, storing the logic and of all the badpixel masks created for each acs image in the set.

For more information on the science applications of the static mask task, see the DrizzlePac Handbook

Parameters

input : str, None (Default = None)

A list of images or associations you would like to use to compute the mask.

static : bool (Default = True)

Create a static bad-pixel mask from the data? This mask flags all pixels that deviate by more than a value of *static_sig* sigma below the image median, since these pixels are typically the result of bad pixel oversubtraction in the dark image during calibration.

static_sig : float (Default = 4.0)

The number of sigma below the RMS to use as the clipping limit for creating the static mask.

editpars : bool (Default = False)

Set to *True* if you would like to edit the parameters using the GUI interface.

Examples

These tasks are designed to work together seemlessly when run in the full *AstroDrizzle* interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined *AstroDrizzle* tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full *AstroDrizzle* interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Basic example of how to call static yourself from a python command line, using the default parameters for the task.

```
>>> from drizzlepac import staticMask
>>> staticMask.createMask('*flt.fits')
```

drizzlepac.staticMask.**createStaticMask**(*imageObjectList=[]*, *configObj=None*, *procSteps=None*)

drizzlepac.staticMask.**getHelpAsString**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

drizzlepac.staticMask.**help**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.staticMask.**run**(*configObj*)

CHAPTER

FIVE

STEP 3: SUBTRACTING THE SKY

This step measures, subtracts and/or equalizes the sky from each input image while recording the subtracted value in the image header.

Authors

Christopher Hanley, Megan Sosey, Mihai Cara

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

`drizzlepac.sky.`**`sky`**(*input=None*, *outExt=None*, *configObj=None*, *group=None*, *editpars=False*, ***inputDict*)

Function for computing and subtracting (or equalizing/matching) the backgroud in input images. The algorithm for sky subtraction can be selected through the *skymethod* parameter. This function will update the `MDRIZSKY` keyword in the headers of the input files.

Sky subtraction is generally recommended for optimal flagging and removal of CR's when the sky background is more than a few electrons. However, some science applications may require the sky to not be removed, allowing for the final drizzle step to be performed with no sky subtraction. If you turn off sky subtraction, you should also set drizzle.pixfrac to 1, otherwise variations in sky between images will add noise to your data.

In addition to the "pure" sky computation, this task can be used for sky "equalization", that is, it can match sky values in the images that are part of a mosaic.

For cameras with multiple detectors (such as ACS/WFC, WFPC2, or WFC3), the sky values in each exposure are first measured separately for the different detectors. These different values are then compared, and the lowest measured sky value is used as the estimate for all of the detectors for that exposure. This is based on the premise that for large extended or bright targets, the pixel intensity distribution in one or more of the detectors may be significantly skewed toward the bright end by the target itself, thereby overestimating the sky on that detector. If the other detector is less affected by such a target, then its sky value will be lower, and can therefore also be substituted as the sky value for the detector with the bright source.

For more information on the science applications of the sky task, see the DrizzlePac Handbook:.

Parameters

input : str or list of str (Default = None)

A python list of image filenames, or just a single filename.

outExt : str (Default = None)

The extension of the output image. If the output already exists then the input image is overwritten.

configObj : configObject (Default = None)

An instance of *configObject*

group : int (Default = None)

The group of the input image.

editpars : bool (Default = False)

A parameter that allows user to edit input parameters by hand in the GUI.

inputDict : dict, optional

An optional list of parameters specified by the user.

Note: These are parameters that *configObj* should contain by default. These parameters can be altered on the fly using the *inputDict*. If *configObj* is set to None and there is no *inputDict* information, then the values for the parameters will be pulled from the default configuration files for the task. These are the same configuration files that are referenced in the TEAL parameter interface GUI. If you wish to edit these parameters by hand in the GUI, simply set editpars=True.

Table of optional parameters that should be in *configobj* and can also be specified in *inputDict*.

Name	Definition
skyuser	'KEYWORD in header which indicates a sky subtraction value to use'.
skymethod	'Sky computation method'
skysub	'Perform sky subtraction?'
skywidth	'Bin width of histogram for sampling sky statistics (in sigma)'
skystat	'Sky correction statistics parameter'
skylower	'Lower limit of usable data for sky (always in electrons)'
skyupper	'Upper limit of usable data for sky (always in electrons)'
skyclip	'Number of clipping iterations'
skylsigma	'Lower side clipping factor (in sigma)'
skyusigma	'Upper side clipping factor (in sigma)'
sky-mask_cat	'Catalog file listing image masks'
use_static	'Use static mask for skymatch computations?'
sky_bits	'Bit flags for identifying bad pixels in DQ array'
skyuser	'KEYWORD indicating a sky subtraction value if done by user'
skyfile	'Name of file with user-computed sky values'
in_memory	'Optimize for speed or for memory use'

These optional parameters are described in more detail below in the "Other Parameters" section.

Returns

None : The input file's primary headers is updated with the computed sky value.

Other Parameters

skysub : bool (Default = Yes)

Turn on or off sky subtraction on the input data. When *skysub* is set to *no*, then *skyuser* field will be enabled and if user specifies a header keyword showing the sky value in the image, then that value will be used for CR-rejection but it will not be subtracted from the (drizzled) image data. If user sets *skysub* to *yes* then *skyuser* field will be disabled (and if it is not empty - it will be ignored) and user can use one of the methods available through the *skymethod* parameter to compute the sky or provide a file (see *skyfile* parameter) with values that should be subtracted from (single) drizzled images.

skymethod : {'localmin', 'globalmin+match', 'globalmin', 'match'}, optional (Default = 'localmin')

Select the algorithm for sky computation:

- **'localmin'**: compute a common sky for all members of *an exposure* (see NOTES below). For a typical use, it will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in an input image, and it will subtract the previously found minimum sky value from all chips (marked for sky subtraction) in that image. This process is repeated for each input image.

Note: This setting is recommended when regions of overlap between images are dominated by "pure" sky (as opposite to extended, diffuse sources).

Note: This is similar to the "skysub" algorithm used in previous versions of astrodrizzle.

- **'globalmin'**: compute a common sky value for all members of *all exposures* (see NOTES below). It will compute sky values for each chip/image extension (marked for sky subtraction in the `input` parameter) in **all** input images, find the minimum sky value, and then it will subtract the **same** minimum sky value from **all** chips (marked for sky subtraction) in **all** images. This method *may* useful when input images already have matched background values.

- **'match'**: compute differences in sky values between images in common (pair-wise) sky regions. In this case computed sky values will be relative (delta) to the sky computed in one of the input images whose sky value will be set to (reported to be) 0. This setting will "equalize" sky values between

the images in large mosaics. However, this method is not recommended when used in conjunction with astrodrizzle because it computes relative sky values while *astrodrizzle* needs "measured" sky values for median image generation and CR rejection.

- **'globalmin+match'**: first find a minimum "global" sky value in all input images and then use **'match'** method to equalize sky values between images.

Note: This is the *recommended* setting for images containing diffuse sources (e.g., galaxies, nebulae) covering significant parts of the image.

---

skywidth : float, optional (Default Value = 0.1)

Bin width, in sigma, used to sample the distribution of pixel flux values in order to compute the sky background statistics.

skystat : {'median', 'mode', 'mean'}, optional (Default Value = 'median')

Statistical method for determining the sky value from the image pixel values.

skylower : float, optional (Default Value = INDEF)

Lower limit of usable pixel values for computing the sky. This value should be specified in the units of the input image.

skyupper : float, optional (Default Value = INDEF)

Upper limit of usable pixel values for computing the sky. This value should be specified in the units of the input image.

skyclip : int, optional (Default Value = 5)

Number of clipping iterations to use when computing the sky value.

skylsigma : float, optional (Default Value = 4.0)

Lower clipping limit, in sigma, used when computing the sky value.

skyusigma : float, optional (Default Value = 4.0)

Upper clipping limit, in sigma, used when computing the sky value.

skymask_cat : str, optional (Default Value = '')

File name of a catalog file listing user masks to be used with images.

use_static : bool, optional (Default Value = True)

Specifies whether or not to use static mask to exclude masked image pixels from sky computations.

sky_bits : int, None, optional (Default = 0)

Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for sky computations. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8

flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for sky computations, then *sky_bits* should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Default value (0) will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for sky computations.

Set *sky_bits* to *None* to turn off the use of image's DQ array for sky computations.

Note: DQ masks (if used), *will be* combined with user masks specified in the input @-file.

---

skyfile : str, optional (Default Value = '')

Name of file containing user-computed sky values to be used with each input image. This ASCII file should only contain 2 columns: image filename in column 1 and sky value in column 2. The sky value should be provided in units that match the units of the input image and for multi-chip images, the same value will be applied to all chips.

skyuser : str (Default = '')

Name of header keyword which records the sky value already subtracted from the image by the user. The *skyuser* parameter is ignored when *skysub* is set to *yes*.

Note: When *skysub*'='*no* and *skyuser* field is empty, then *AstroDrizzle* will assume that sky background is 0.0 for the purpose of cosmic-ray rejection.

---

in_memory : bool, optional (Default Value = False)

Specifies whether to optimize execution for speed (maximum memory usage) or use a balanced approach in which a minimal amount of image data is kept in memory and retrieved from disk as needed. The default setting is recommended for most systems.

Notes

`sky` provides new algorithms for sky value computations and enhances previously available algorithms used by, e.g., Astrodrizzle.

First, the standard sky computation algorithm (see *skymethod* = `'localmin'`) was upgraded to be able to use DQ flags and user supplied masks to remove "bad" pixels from being used for sky statistics computations.

Second, two new methods have been introduced: `'globalmin'` and `'match'`, as well as a combination of the two – `'globalmin+match'`.

- The `'globalmin'` method computes the minimum sky value across *all* chips in *all* input images. That sky value is then considered to be the background in all input images.
- The `'match'` algorithm is somewhat similar to the traditional sky subtraction method (*skymethod*=`'localmin'`) in the sense that it measures the sky indipendently in input images (or detector chips). The major differences are that, unlike the traditional method,
  1. `'match'` algorithm computes *relative* sky values with regard to the sky in a reference image chosen from the input list of images; *and*
  2. Sky statistics is computed only in the part of the image that intersects other images.

  This makes `'match'` sky computation algorithm particularly useful for "equalizing" sky values in large mosaics in which one may have only (at least) pair-wise intersection of images without having a common intersection region (on the sky) in all images.

  The *'match'* method works in the following way: for each pair of intersecting images, an equation is written that requires that average surface brightness in the overlapping part of the sky be equal in both images. The final system of equations is then solved for unknown background levels.

  Warning: Current algorithm is not capable of detecting cases when some groups of intersecting images (from the input list of images) do not intersect at all other groups of intersecting images (except for the simple case when *single* images do not intersect any other images). In these cases the algorithm will find equalizing sky values for each group. However since these groups of images do not intersect each other, sky will be matched only within each group and the "inter-group" sky mismatch could be significant.
  Users are responsible for detecting such cases and adjusting processing accordingly.

  Warning: Because this method computes *relative sky values* compared to a reference image (which will have its sky value set to 0), the sky values computed with this method usually are smaller than the "absolute" sky values computed, e.g., with the `'localmin'` algorithm. Since astrodrizzle expects "true" (as opposite to *relative*) sky values in order to correctly compute the median image or to perform cosmic-ray detection, this algorithm in not recommended to be used *alone* for sky computations to be used with *astrodrizzle*.
  For the same reason, IVM weighting in *astrodrizzle* should **not** be used with `'match'` method: sky values reported in `MDRIZSKY` header keyword will be relative sky values (sky offsets) and derived weights will be incorrect.

- The `'globalmin+match'` algorithm combines `'match'` and `'globalmin'` methods in order to overcome the limitation of the `'match'` method described in the note above: it uses `'globalmin'` algorithm to find a baseline sky value common to all input images and the `'match'` algorithm to "equalize" sky values in the mosaic. Thus, the sky value of the "reference" image will be equal to the baseline sky value (instead of 0 in `'match'` algorithm alone)

making this method acceptable for use in conjunction with *astrodrizzle*.

Glossary:

Exposure – a *subset* of FITS image extensions in an input image that correspond to different chips in the detector used to acquire the image. The subset of image extensions that form an exposure is defined by specifying extensions to be used with input images (see parameter *input*).

See help for `skypac.parseat.parse_at_line` for details on how to specify image extensions.

Footprint – the outline (edge) of the projection of a chip or of an exposure on the celestial sphere.

Note:

- Footprints are managed by the `SphericalPolygon` class.
- Both footprints *and* associated exposures (image data, WCS information, and other header information) are managed by the `SkyLine` class.
- Each `SkyLine` object contains one or more `SkyLineMember` objects that manage both footprints *and* associated *chip* data that form an exposure.

Remarks:

- *sky* works directly on *geometrically distorted* flat-fielded images thus avoiding the need to perform an additional drizzle step to perform distortion correction of input images.

  Initially, the footprint of a chip in an image is aproximated by a 2D planar rectangle representing the borders of chip's distorted image. After applying distortion model to this rectangle and progecting it onto the celestial sphere, it is approximated by spherical polygons. Footprints of exposures and mosaics are computed as unions of such spherical polygons while overlaps of image pairs are found by intersecting these spherical polygons.

Limitations and Discussions:

Primary reason for introducing "sky match" algorithm was to try to equalize the sky in large mosaics in which computation of the "absolute" sky is difficult due to the presence of large diffuse sources in the image. As discussed above, *sky* accomplishes this by comparing "sky values" in a pair of images in the overlap region (that is common to both images). Quite obviously the quality of sky "matching" will depend on how well these "sky values" can be estimated. We use quotation marks around *sky values* because for some image "true" background may not be present at all and the measured sky may be the surface brightness of large galaxy, nebula, etc.

Here is a brief list of possible limitations/factors that can affect the outcome of the matching (sky subtraction in general) algorithm:

- Since sky subtraction is performed on *flat-fielded* but *not distortion corrected* images, it is important to keep in mind that flat-fielding is performed to obtain uniform surface brightness and not flux. This distinction is important for images that have not been distortion corrected. As a consequence, it is advisable that point-like sources be masked through the

user-supplied mask files. Alternatively, one can use *upper* parameter to limit the use of bright objects in sky computations.

- Normally, distorted flat-fielded images contain cosmic rays. This algorithm does not perform CR cleaning. A possible way of minimizing the effect of the cosmic rays on sky computations is to use clipping (*nclip* > 0) and/or set *upper* parameter to a value larger than most of the sky background (or extended source) but lower than the values of most CR pixels.

- In general, clipping is a good way of eliminating "bad" pixels: pixels affected by CR, hot/dead pixels, etc. However, for images with complicated backgrounds (extended galaxies, nebulae, etc.), affected by CR and noise, clipping process may mask different pixels in different images. If variations in the background are too strong, clipping may converge to different sky values in different images even when factoring in the "true" difference in the sky background between the two images.

- In general images can have different "true" background values (we could measure it if images were not affected by large diffuse sources). However, arguments such as *lower* and *upper* will apply to all images regardless of the intrinsic differences in sky levels.

How to use the tasks stand alone interface in your own scripts:

These tasks are designed to work together seemlessly when run in the full *AstroDrizzle* interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined *AstroDrizzle* tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full *AstroDrizzle* interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

Basic example of how to call sky yourself from a python command line, this example will use the default parameter settings and subtract a sky value from each `*flt.fits` image in the current directory, saving the output file with the extension of "mysky":

```
>>> from drizzlepac import sky
>>> sky.sky('*flt.fits',outExt='mysky')
```

`drizzlepac.sky.`**`help`**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

CHAPTER

SIX

STEP 4 AND 8: DRIZZLING THE IMAGES

The operation of drizzling each input image needs to be performed twice during processing:

- single drizzle step: this initial step drizzles each image onto the final output WCS as separate images
- final drizzle step: this step produces the final combined image based on the cosmic-ray masks determined by *MultiDrizzle*

Interfaces to main drizzle functions.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.adrizzle.**buildDrizParamDict**(*configObj*, *single=True*)

drizzlepac.adrizzle.**create_output**(*filename*)

drizzlepac.adrizzle.**do_driz**(*insci*, *input_wcs*, *inwht*, *output_wcs*, *outsci*, *outwht*, *outcon*, *expin*, *in_units*, *wt_scl*, *wcslin_pscale=1.0*, *uniqid=1*, *pixfrac=1.0*, *kernel='square'*, *fillval='INDEF'*, *stepsize=10*, *wcsmap=None*)

Core routine for performing 'drizzle' operation on a single input image All input values will be Python objects such as ndarrays, instead of filenames. File handling (input and output) will be performed by calling routine.

drizzlepac.adrizzle.**drizFinal**(*imageObjectList*, *output_wcs*, *configObj*, *build=None*, *wcsmap=None*, *procSteps=None*)

drizzlepac.adrizzle.**drizSeparate**(*imageObjectList*, *output_wcs*, *configObj*, *wcsmap=None*, *procSteps=None*)

drizzlepac.adrizzle.**drizzle**(*input*, *outdata*, *wcsmap=None*, *editpars=False*, *configObj=None*, ***input_dict*)

Each input image gets drizzled onto a separate copy of the output frame. When stacked, these copies would correspond to the final combined product. As separate images, they allow for treatment of each input image separately in the undistorted, final WCS system. These images provide the information necessary for refining image registration for each of the input images. They also provide the images

that will be succeedingly combined into a median image and then used for the subsequent blot and cosmic ray detection steps.

Aside from the input parameters, this step requires:

•valid input images with SCI extensions

•valid distortion coefficients tables

•any optional secondary distortion correction images

•numpy object (in memory) for static mask

This step produces:

•singly drizzled science image (simple FITS format)

•singly drizzled weight images (simple FITS format)

These images all have the same WCS based on the original input parameters and those provided for this step; specifically, output shape, pixel size, and orientation, if any have been specified at all.

For more information on the science applications of the drizzle task, see the AstroDrizzle Handbook

Other Parameters

driz_separate : bool (Default = No)

This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the same WCS as the final combined output frame. These images are used to create the median image, needed for cosmic ray rejection.

driz_sep_kernel : {'square', 'point', 'gaussian', 'turbo', 'tophat', 'lanczos3'} (Default = 'turbo')

Used for the initial separate drizzling operation only, this parameter specifies the form of the kernel function used to distribute flux onto the separate output images. The current options are:

•**square**: original classic drizzling kernel

•**point**: this kernel is a point so each input pixel can only contribute to the single pixel that is closest to the output position. It is equivalent to the limit as `pixfrac` -> 0, and is very fast.

•**gaussian**: this kernel is a circular gaussian with a FWHM equal to the value of `pixfrac`, measured in input pixels.

•**turbo**: this is similar to kernel="square" but the box is always the same shape and size on the output grid, and is always aligned with the X and Y axes. This may result in a significant speed increase.

•**tophat**: this kernel is a circular "top hat" shape of width `pixfrac`. It effects only output pixels within a radius of `pixfrac`/2 from the output position.

•**lanczos3**: a Lanczos style kernel, extending a radius of 3 pixels from the center of the detection. The Lanczos kernel is a damped and bounded form

of the "sinc" interpolator, and is very effective for resampling single images when `scale`=`pixfrac`=1. It leads to less resolution loss than other kernels, and typically results in reduced correlated noise in outputs.

> **Warning:** The `'lanczos3'` kernel tends to result in much slower processing as compared to other kernel options. This option should never be used for `pixfrac`!=1.0, and is not recommended for `scale` != 1.0.

The default for this step is **"turbo"** since it is much faster than **"square"**, and it is quite satisfactory for the purposes of generating the median image. More information about the different kernels can be found in the help file for the drizzle task.

driz_sep_wt_scl : float (Default = exptime)

This parameter specifies the weighting factor for input image. If *driz_sep_wt_scl*=`exptime`, then the scaling value will be set equal to the exposure time found in the image header. The use of the default value is recommended for producing optimal behavior for most scenarious. It is possible to set *wt_scl*='expsq' for weighting by the square of the exposure time, which is optimal for read-noise dominated images.

driz_sep_pixfrac : float (Default = 1.0)

Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If `pixfrac` is set to less than 0.001, the kernel parameter will be reset to 'point' for more efficient processing. In the step of drizzling each input image onto a separate output image, the default value of 1.0 is best in order to ensure that each output drizzled image is fully populated with pixels from the input image. For more information, see the help for the *drizzle* task.

driz_sep_fillval : int or INDEF (Default = INDEF)

Value to be assigned to output pixels that have zero weight, or that receive flux from any input pixels during drizzling. This parameter corresponds to the *fillval* parameter of the *drizzle* task. If the default of `INDEF` is used, and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (e.g. 0), then these pixels will be set to that value.

driz_sep_bits : int (Default = 0)

Integer sum of all the DQ bit values from the input image's DQ array that should be considered 'good' when building the weighting mask. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of *AstroDrizzle*, by adding the value 4096 for ACS and WFPC2 data. Please see the section on Selecting the *Bits* Parameter for a more detailed discussion.

driz_sep_wcs : bool (Default = No)

Define custom WCS for seperate output images?

driz_sep_refimage : str (Default = '')

Reference image from which a WCS solution can be obtained.

driz_sep_rot : float (Default = INDEF)

Position Angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of `INDEF` specifies that the images will not be rotated, but will instead be drizzled in the default orientation for the camera with the x and y axes of the drizzled image corresponding approximately to the detector axes. This conserves disk space, as these single drizzled images are only used in the intermediate step of creating a median image.

driz_sep_scale : float (Default = INDEF)

Linear size of the output pixels in arcseconds/pixel for each separate drizzled image (used in creating the median for cosmic ray rejection). The default value of `INDEF` specifies that the undistorted pixel scale for the first input image will be used as the pixel scale for all the output images.

driz_sep_outnx : int (Default = INDEF)

Size, in pixels, of the X axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_outny : int (Default = INDEF)

Size, in pixels, of the Y axis in the output images that each input will be drizzled onto. If no value is specified, the smallest size that can accommodate the full dithered field will be used.

driz_sep_ra : float (Default = INDEF)

Right ascension (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

driz_sep_dec : float (Default = INDEF)

Declination (in decimal degrees) specifying the center of the output image. If this value is not designated, the center will automatically be calculated based on the distribution of image dither positions.

Notes

These tasks are designed to work together seemlessly when run in the full *AstroDrizzle* interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined *AstroDrizzle* tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full *AstroDrizzle* interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

There are two user interface function for this task, one to allow you to create seperately drizzled images of each image in your list and the other to create one single output drizzled image, which is the combination of all of them:

```
    def drizSeparate(imageObjectList,output_wcs,configObj,wcsmap=wcs_functions.WCSMap)
    def drizFinal(imageObjectList, output_wcs, configObj,build=None,wcsmap=wcs_functions.W
    if configObj[single_step]['driz_separate']:
        drizSeparate(imgObjList,outwcs,configObj,wcsmap=wcsmap)
    else:
        drizFinal(imgObjList,outwcs,configObj,wcsmap=wcsmap)
```

Examples

Basic example of how to call static yourself from a python command line, using the default parameters for the task.

```
    >>> from drizzlepac import adrizzle
```

drizzlepac.adrizzle.**getHelpAsString**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called `__taskname__.help`

drizzlepac.adrizzle.**get_data**(*filename*)

drizzlepac.adrizzle.**help**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.adrizzle.**interpret_maskval**(*paramDict*)
Apply logic for interpreting final_maskval value...

drizzlepac.adrizzle.**mergeDQarray**(*maskname*, *dqarr*)
Merge static or CR mask with mask created from DQ array on-the-fly here.

drizzlepac.adrizzle.**run**(*configObj*, *wcsmap=None*)
Interface for running *wdrizzle* from TEAL or Python command-line.

This code performs all file `I/O` to set up the use of the drizzle code for a single exposure to replicate the functionality of the original *wdrizzle*.

drizzlepac.adrizzle.**run_driz**(*imageObjectList*, *output_wcs*, *paramDict*, *single*, *build*, *wcsmap=None*)
Perform drizzle operation on input to create output. The input parameters originally was a list of dictionaries, one for each input, that matches the primary parameters for an `IRAF` *drizzle* task.

This method would then loop over all the entries in the list and run *drizzle* for each entry.

Parameters required for input in paramDict:
build,single,units,wt_scl,pixfrac,kernel,fillval, rot,scale,xsh,ysh,blotnx,blotny,outnx,outny,data

drizzlepac.adrizzle.**run_driz_chip**(*img*, *chip*, *output_wcs*, *outwcs*, *template*, *paramDict*, *single*, *doWrite*, *build*, *_versions*, *_numctx*, *_nplanes*, *_numchips*, *_outsci*, *_outwht*, *_outctx*, *_hdrlist*, *wcsmap*)

Perform the drizzle operation on a single chip. This is separated out from *run_driz_img* so as to keep together the entirety of the code which is inside the loop over chips. See the *run_driz* code for more documentation.

drizzlepac.adrizzle.**run_driz_img**(*img*, *chiplist*, *output_wcs*, *outwcs*, *template*, *paramDict*, *single*, *num_in_prod*, *build*, *_versions*, *_numctx*, *_nplanes*, *chipIdxCopy*, *_outsci*, *_outwht*, *_outctx*, *_hdrlist*, *wcsmap*)

Perform the drizzle operation on a single image. This is separated out from *run_driz* so as to keep together the entirety of the code which is inside the loop over images. See the *run_driz* code for more documentation.

drizzlepac.adrizzle.**updateInputDQArray**(*dqfile*, *dq_extn*, *chip*, *crmaskname*, *cr_bits_value*)

CHAPTER

SEVEN

STEP 5: CREATE A MEDIAN IMAGE

A median image gets generated from the stack of undistorted single drizzle images. Create a median image from the singly drizzled images.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.createMedian.**createMedian**(*imgObjList*, *configObj*, *procSteps=None*)

Top-level interface to createMedian step called from top-level MultiDrizzle.

This function parses the input parameters then calls the *_median()* function to median-combine the input images into a single image.

drizzlepac.createMedian.**getHelpAsString**(*docstring=False*, *show_ver=True*)

return useful help from a file in the script directory called __taskname__.help

drizzlepac.createMedian.**help**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.createMedian.**median**(*input=None*, *configObj=None*, *editpars=False*, ***inputDict*)

The singly drizzled science images are combined to create a single median image. This median combination gets performed section-by-section from the input single drizzled images. Each section corresponds to a contiguous set of lines from each image taking up no more than 1Mb in memory, such that combining 10 input images would only require 10Mb for these sections. The goal of this step is to establish an estimate for what the fully cleaned image should look like in order to enable better bad pixel detection, in addition to improving the alignment of the image stack. Creating a median image from the aligned and undistorted input images allows for a statistical rejection of bad pixels.

The final median image serves as the only output from this step.

For more information on the science applications of the `createMedian` task, see the DrizzlePac Handbook.

Parameters

input : str or list of str (Default = None)

A python list of drizzled image filenames, or just a single filename.

configObj : configObject (Default = None)

An instance of *configObject* which overrides default parameter settings.

editpars : bool (Default = False)

A parameter that allows user to edit input parameters by hand in the GUI. True to use the GUI to edit parameters.

inputDict : dict, optional

An optional list of parameters specified by the user, which can also be used to override the defaults.

Other Parameters

median : bool (Default = No)

This parameter specifies whether or not to create a median image. This median image will be used as the comparison 'truth' image :in the cosmic ray rejection step.

median_newmasks : bool (Default = Yes)

This parameter specifies whether or not new mask files will be created when the median image is created. These masks are generated from weight files previously produced by the "driz_separate" step, and contain all bad pixel information used to exclude pixels when calculating the median. Generally this step should be set to "Yes", unless for some reason, it is desirable to include bad pixel information when generating the median.

combine_maskpt : float (Default = 0.7)

Percentage of weight image values, below which the are flagged.

combine_type : str {'average', 'median', 'sum', 'minmed'} (Default = 'minmed')

This parameter defines the method that will be used to create the median image. The 'average', 'median', and 'sum' options set the calculation type when running 'numcombine', a numpy method for median-combining arrays to create the median image. The "minmed" option will produce an image that is generally the same as the median, except in cases where the median is significantly higher than the minimum good pixel value. In this case, "minmed" will choose the minimum value. The sigma thresholds for this decision are provided by the "combine_nsigma" parameter. However, as the "combine_nsigma" parameter does not adjust for the larger probability of a single "nsigma" event with a greater number of images, "minmed" will bias the comparison image low for a large number of images. "minmed" is highly

recommended for three images, and is good for four to six images, but should be avoided for ten or more images.

A value of 'median' is the recommended method for a large number of images, and works equally well as minmed down to approximately four images. However, the user should set the "combine_nhigh" parameter to a value of 1 when using "median" with four images, and consider raising this parameter's value for larger numbers of images. As a median averages the two inner values when the number of values being considered is even, the user may want to keep the total number of images minus "combine_nhigh" odd when using "median".

combine_nsigma : float (Default = '4 3')

This parameter defines the sigmas used for accepting minimum values, rather than median values, when using the 'minmed' combination method. If two values are specified the first value will be used in the initial choice between median and minimum, while the second value will be used in the "growing" step to reject additional pixels around those identified in the first step. If only one value is specified, then it is used in both steps.

combine_nlow : int (Default = 0)

This parameter sets the number of low value pixels to reject automatically during image combination.

combine_nhigh : int (Default = 0)

This parameter sets the number of high value pixels to reject automatically during image combination.

combine_lthresh : float (Default = INDEF)

Sets the lower threshold for clipping input pixel values during image combination. This value gets passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_hthresh : float (Default = INDEF)

This parameter sets the upper threshold for clipping input pixel values during image combination. The value for this parameter is passed directly to 'imcombine' for use in creating the median image. If the parameter is set to "None", no thresholds will be imposed.

combine_grow : int (Default = 1)

Width, in pixels, beyond the limit set by the rejection algorithm being used, for additional pixels to be rejected in an image. This parameter is used to set the 'grow' parameter in 'imcombine' for use in creating the median image.

combine_bufsize : float (Default = None)

Size of buffer, in Mb, to use when reading in each section of each input image. The default buffer size is 1Mb. The larger the buffer size, the fewer

times the code needs to open each input image and the more memory will be required to create the median image. A larger buffer can be helpful when using compression, since slower copies need to be made of each set of rows from each input image instead of using memory-mapping.

Examples

For *createMedian*, the user interface function is *median*:

```
>>> from drizzlepac import createMedian
>>> createMedian.median('*flt.fits')
```

drizzlepac.createMedian.**run**(*configObj*)

CHAPTER

EIGHT

STEP 6: BLOTTING THE MEDIAN IMAGE

In this step the median image now gets blotted back to create median-cleaned images which can be compared directly with each input image to identify cosmic-rays.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

`drizzlepac.ablot.`**`blot`**(*data*, *outdata*, *configObj=None*, *wcsmap=wcs_functions.WCSMap*, *editpars=False*, ***input_dict*)

The median image is the combination of the WCS aligned input images that have already had the distortion model applied. Taking the median of the aligned images allows for a statistical rejection of bad pixels from the image stack. The resulting median image can then be input for the blot task with the goal of creating 'cleaned' versions of the input images at each of their respective dither locations. These "blotted" images can then be directly compared to the original distorted input images for detection of image artifacts (i.e. bad-pixels, hot pixels, and cosmic-rays) whose locations will be saved to the output badpixel masks.

Aside from the input parameters, this step only requires opening the single median image created from all the input images. A distorted version of the median image corresponding to each input 'chip' (extension) is written as output from this step as separate simple FITS images.

For more information on the science applications of the blot task, see the DrizzlePac Handbook

Parameters

data : str

Input distortion-corrected (median or drizzled) image to be used as the source for creating blotted images.

reference : str

Filename of image to read to define the blotted WCS; image with distortion to be matched by output blotted image.

outdata : str

Filename for output blotted image.

coeffs : bool (Default Value = True)

This parameters specifies whether or not to use the header-based distortion coefficients when creating the blotted, distorted image. If False, no distortion will be applied at all, effectively working as a cut-out operation.

interp : str{'nearest', 'linear', 'poly3', 'poly5', 'sinc'} (Default = 'poly5')

This parameter defines the method of interpolation to be used when blotting drizzled images back to their original WCS solution. Valid options include:

- **nearest**: Nearest neighbor
- **linear**: Bilinear interpolation in x and y
- **poly3**: Third order interior polynomial in x and y
- **poly5**: Fifth order interior polynomial in x and y
- **sinc**: Sinc interpolation (accurate but slow)

The 'poly5' interpolation method has been chosen as the default because it is relatively fast and accurate.

If 'sinc' interpolation is selected, then the value of the parameter for *blot_sinscl* will be used to specify the size of the sinc interpolation kernel.

sinscl : float (Default Value = 1.0)

Size of the sinc interpolation kernel in pixels.

stepsize : int (Default Value = 10)

Number of pixels for WCS interpolation. The distortion model will be sampled exactly and completely every *stepsize* pixel with bi-linear interpolation being used to compute the distortion for intermediate pixels. This optimization speeds up the computation significantly when *stepsize* >> 1 at the expense of interpolation errors for intermediate pixels.

addsky : bool (Default Value = Yes)

Add back a sky value using the `MDRIZSKY` value from the header. If 'Yes' (*True*), the *blot_skyval* parameter is ignored.

skyval : float (Default Value = 0.0)

This is a user-specified custom sky value to be added to the blot image. This is only used if *blot_addsky* is 'No' (*False*).

in_units : str{'cps', 'counts'} (Default Value= 'cps')

Units of input (drizzled) image. Valid options are **'cps'** and **'counts'**.

out_units : str{'cps', 'counts'} (Default Value = 'counts')

Units of the ouput (blotted) image. Valid options are **'cps'** and **'counts'**.

expkey : str (Default Value = 'exptime)

Name of keyword to use to extract exposure time value, which will be used to scale the blotted image to the final output flux values when *out_units* is set to **counts**.

expout : str or float (Default Value = 'input')

Value of exposure time to use in scaling the output blotted image when *out_units* is set to **counts**. If set to **'input'**, the value will be read in from the input image header keyword specified by *expkey*.

Note: The following parameters, when set, will override any value determined from *refimage* if a reference image was specified.

---

outscale : float,optional

Absolute size of output pixels in arcsec/pixel

orient : float

Orientation of output (PA of Y axis, N through E)

raref : float

RA of reference point on output image(CRVAL1,degrees)

decref : float

Dec of reference point on output image (CRVAL2, degrees)

xrefpix : float

Reference pixel X position on output (CRPIX1)

yrefpix : float

Reference pixel Y position on output (CRPIX2)

outnx : float

Size of output image's X-axis (pixels)

outny : float

Size of output image's Y-axis (pixels)

Notes

These tasks are designed to work together seemlessly when run in the full *AstroDrizzle* interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined *AstroDrizzle* tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full *AstroDrizzle* interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

1. Basic example of how to call *blot* yourself from a python command line, using the default parameter settings:

```
>>> from drizzlepac import ablot
>>> ablot.blot()
```

2. Creation of a blotted image from the products generated by running the *astrodrizzle* task can be done for the median image "adriz_aligned_wcs_f814w_med.fits" to re-create the (SCI,1) chip from "j8c0d1bwq_flc.fits" using:

```
>>> from drizzlepac import ablot
>>> from stsci.tools import teal
>>> blotobj = teal.load('ablot') # get default values
>>> ablot.blot('adriz_aligned_wcs_f814w_med.fits','j8c0d1bwq_flc.fits[sci,1]',
'aligned_f814w_sci1_blot.fits',configObj=blotobj)
```

or

```
>>> a = teal.teal('ablot')
# set data = adriz_aligned_wcs_f814w_med.fits
# set reference = j8c0d1bwq_flc.fits[sci,1]
# set outdata = aligned_f814w_sci1_blot.fits
```

drizzlepac.ablot.**buildBlotParamDict**(*configObj*)

drizzlepac.ablot.**do_blot**(*source*, *source_wcs*, *blot_wcs*, *exptime*, *coeffs=True*, *interp='poly5'*, *sinscl=1.0*, *stepsize=10*, *wcsmap=None*)

Core functionality of performing the 'blot' operation to create a single blotted image from a single source image. All distortion information is assumed to be included in the WCS specification of the 'output' blotted image given in 'blot_wcs'.

This is the simplest interface that can be called for stand-alone use of the blotting function.

Parameters

source

Input numpy array of undistorted source image in units of 'cps'.

source_wcs

HSTWCS object representing source image distortion-corrected WCS.

blot_wcs

(py)wcs.WCS object representing the blotted image WCS.

exptime

exptime to use for scaling output blot image. A value of 1 will result in output blot image in units of 'cps'.

coeffs

Flag to specify whether or not to use distortion coefficients associated with blot_wcs. If False, do not apply any distortion model.

interp

Form of interpolation to use when blotting pixels. Valid options:

```
"nearest","linear","poly3", "poly5"(default), "spline3", "sinc"
```

sinscl

Scale for sinc interpolation kernel (in output, blotted pixels)

stepsize

Number of pixels for WCS interpolation

wcsmap

Custom mapping class to use to provide transformation from drizzled to blotted WCS. Default will be to use *drizzlepac.wcs_functions.WCSMap*.

drizzlepac.ablot.**getHelpAsString**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

drizzlepac.ablot.**help**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.ablot.**run**(*configObj*, *wcsmap=None*)
Run the blot task based on parameters provided interactively by the user.

drizzlepac.ablot.**runBlot**(*imageObjectList*, *output_wcs*, *configObj={}*, *wcsmap=wcs_functions.WCSMap*, *procSteps=None*)

drizzlepac.ablot.**run_blot**(*imageObjectList*, *output_wcs*, *paramDict*, *wcsmap=wcs_functions.WCSMap*)
Perform the blot operation on the list of images.

CHAPTER

NINE

STEP 7: COSMIC-RAY IDENTIFICATION

The cosmic rays and bad pixels are now identified by comparing the input images with the associated blotted,median-cleaned images created. Mask blemishes in dithered data by comparison of an image with a model image and the derivative of the model image.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.drizCR.**createCorrFile**(*outfile*, *arrlist*, *template*)

Create a _cor file with the same format as the original input image

The DQ array will be replaced with the mask array used to create the _cor file.

drizzlepac.drizCR.**drizCR**(*input=None*, *configObj=None*, *editpars=False*, ***inputDict*)

The blotted median images that are now transformed back into the original reference frame, get compared to the original input images to detect any spurious pixels (which may include pixels impacted by cosmic rays) in each input. Those spurious pixels then get flagged as 'bad' in the output cosmic ray mask files, which get used as input for the final combination so that they do not show up in the final product. The identified bad pixels get flagged by updating the input mask files. Optionally, copies of the original images with the bad pixels removed can be created through the use of the *driz_cr_corr* parameter.

Parameters

input : str or list of str (Default = None)

A python list of blotted median image filenames, or just a single filename.

configObj : configObject (Default = None)

An instance of *configObject* which overrides default parameter settings.

editpars : bool (Default = False)

A parameter that allows user to edit input parameters by hand in the GUI. True to use the GUI to edit parameters.

inputDict : dict, optional

An optional list of parameters specified by the user, which can also be used to override the defaults.

Other Parameters

driz_cr : bool (Default = False)

Perform cosmic-ray detection? If set to `True`, cosmic-rays will be detected and used to create cosmic-ray masks based on the algorithms from *deriv* and *driz_cr*.

driz_cr_corr : bool (Default = False)

Create a cosmic-ray cleaned input image? I set to *True*, a cosmic-ray cleaned `_cor` image will be generated directly from the input image, and a corresponding `_crmask` file will be written to document detected pixels affected by cosmic-rays.

driz_cr_snr : list of floats (Default = '3.5 3.0')

The values for this parameter specify the signal-to-noise ratios for the *driz_cr* task to be used in detecting cosmic rays. See the help file for *driz_cr* for further discussion of this parameter.

driz_cr_grow : int (Default = 1)

The radius, in pixels, around each detected cosmic-ray, in which more stringent detection criteria for additional cosmic rays will be used.

driz_cr_ctegrow : int (Default = 0)

Length, in pixels, of the CTE tail that should be masked in the drizzled output.

driz_cr_scale : str (Default = '1.2 0.7')

Scaling factor applied to the derivative in *driz_cr* when detecting cosmic-rays. See the help file for *driz_cr* for further discussion of this parameter.

Notes

These tasks are designed to work together seemlessly when run in the full *AstroDrizzle* interface. More advanced users may wish to create specialized scripts for their own datasets, making use of only a subset of the predefined AstroDrizzle tasks, or add additional processing, which may be usefull for their particular data. In these cases, individual access to the tasks is important.

Something to keep in mind is that the full *AstroDrizzle* interface will make backup copies of your original files and place them in the `OrIg/` directory of your current working directory. If you are working with the stand alone interfaces, it is assumed that the user has already taken care of backing up their original datafiles as the input file with be directly altered.

Examples

Basic example of how to call drizCR yourself from a python command line using the default parameters for the task.

```
>>> from drizzlepac import drizCR
>>> drizCR.drizCR('*flt.fits')
```

`drizzlepac.drizCR.`**`getHelpAsString`**(*docstring=False*, *show_ver=True*)

return useful help from a file in the script directory called __taskname__.help

drizzlepac.drizCR.**help**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.drizCR.**run**(*configObj*)

drizzlepac.drizCR.**rundrizCR**(*imgObjList*, *configObj*, *procSteps=None*)

drizzlepac.drizCR.**setDefaults**(*configObj={}*)

Return a dictionary of the default parameters which also been updated with the user overrides.

CHAPTER

TEN

UTILITIES

Various utilities get used by *Multidrizzle*, including some to handle WCS interpretation, trailer file generation, output file generation and interpretation of the `MDRIZTAB` reference file.

10.1 Utility Functions

These functions perform various small operations within *Multidrizzle*. A library of utility functions

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class `drizzlepac.util.`**`ProcSteps`**
This class allows MultiDrizzle to keep track of the start and end times of each processing step that gets run as well as computing/reporting the elapsed time for each step.

The code for each processing step must call the 'addStep()' method to initialize the information for that step, then the 'endStep()' method to record the end and elapsed times.

The 'reportTimes()' method can then be used to provide a summary of all the elapsed times and total run time.

`addStep`(*key*)
Add information about a new step to the dict of steps The value 'ptime' is the output from '_ptime()' containing both the formatted and unformatted time for the start of the step.

`endStep`(*key*)
Record the end time for the step.

If key==None, simply record ptime as end time for class to represent the overall runtime since the initialization of the class.

`reportTimes`()
Print out a formatted summary of the elapsed times for all the performed steps.

class `drizzlepac.util.`**`WithLogging`**

drizzlepac.util.**applyUserPars_steps**(*configObj*, *input_dict*, *step='3a'*)
Apply logic to turn on use of user-specified output WCS if user provides any parameter on command-line regardless of how final_wcs was set.

drizzlepac.util.**atfile_ivm**(*filename*)
Return the filename of the IVM file which is assumed to be the second word in the atfile the user gave.

drizzlepac.util.**atfile_sci**(*filename*)
Return the filename of the science image which is assumed to be the first word in the atfile the user gave.

drizzlepac.util.**base_taskname**(*taskname*, *packagename=None*)
Extract the base name of the task.

Many tasks in the drizzlepac have "compound" names such as 'drizzlepac.sky'. This function will search for the presence of a dot in the input *taskname* and if found, it will return the string to the right of the right-most dot. If a dot is not found, it will return the input string.

Parameters

taskname : str, None

Full task name. If it is *None*, base_taskname will return *None*.

packagename : str, None (Default = None)

Package name. It is assumed that a compound task name is formed by concatenating *packagename* + '.' + *taskname*. If *packagename* is not *None*, base_taskname will check that the string to the left of the right-most dot matches *packagename* and will raise an *AssertionError* if the package name derived from the input *taskname* does not match the supplied *packagename*. This is intended as a check for discrepancies that may arise during the development of the tasks. If *packagename* is *None*, no such check will be performed.

Raises

AssertionError

Raised when package name derived from the input *taskname* does not match the supplied *packagename*

drizzlepac.util.**check_blank**(*cvar*)
Converts blank value (from configObj?) into a value of None.

drizzlepac.util.**computeRange**(*corners*)
Determine the range spanned by an array of pixel positions.

drizzlepac.util.**compute_texptime**(*imageObjectList*)
Add up the exposure time for all the members in the pattern, since 'drizzle' doesn't have the necessary information to correctly set this itself.

drizzlepac.util.**countImages**(*imageObjectList*)

drizzlepac.util.**count_sci_extensions**(*filename*)
Return the number of SCI extensions and the EXTNAME from a input MEF file.

drizzlepac.util.**createFile**(*dataArray=None*, *outfile=None*, *header=None*)
Create a simple fits file for the given data array and header. Returns either the FITS object in-membory when outfile==None or None when the FITS file was written out to a file.

drizzlepac.util.**displayBadRefimageWarningBox**(*display=True*, *parent=None*)
Displays a warning box for the 'input' parameter.

drizzlepac.util.**displayEmptyInputWarningBox**(*display=True*, *parent=None*)
Displays a warning box for the 'input' parameter.

drizzlepac.util.**displayMakewcsWarningBox**(*display=True*, *parent=None*)
Displays a warning box for the 'makewcs' parameter.

drizzlepac.util.**end_logging**(*filename=None*)
Close log file and restore system defaults.

drizzlepac.util.**findWCSExtn**(*filename*)
Return new filename with extension that points to an extension with a valid WCS.

Returns

extnum : str, None

Value of extension name as a string either as provided by the user or based on the extension number for the first extension which contains a valid HSTWCS object. Returns None if no extension can be found with a valid WCS.

Notes

The return value from this function can be used as input to
create another HSTWCS with the syntax:

```
`HSTWCS('{}[{}]'.format(filename,extnum))
```

drizzlepac.util.**findrootname**(*filename*)
Return the rootname of the given file.

drizzlepac.util.**getConfigObjPar**(*configObj*, *parname*)
Return parameter value without having to specify which section holds the parameter.

drizzlepac.util.**getDefaultConfigObj**(*taskname*, *configObj*, *input_dict={}*, *loadOnly=True*)
Return default configObj instance for task updated with user-specified values from input_dict.

Parameters

taskname : string

Name of task to load into TEAL

configObj : string

The valid values for 'configObj' would be:

```
                None                        - loads last saved user .cfg file
                'defaults'                  - loads task default .cfg file
                name of .cfg file (string)- loads user-specified .cfg file
```

input_dict : dict

Set of parameters and values specified by user to be different from what gets loaded in from the .cfg file for the task

loadOnly : bool

Setting 'loadOnly' to False causes the TEAL GUI to start allowing the user to edit the values further and then run the task if desired.

drizzlepac.util.**getFullParList**(*configObj*)
Return a single list of all parameter names included in the configObj regardless of which section the parameter was stored

drizzlepac.util.**getRotatedSize**(*corners*, *angle*)
Determine the size of a rotated (meta)image.

drizzlepac.util.**getSectionName**(*configObj*, *stepnum*)
Return section label based on step number.

drizzlepac.util.**get_detnum**(*hstwcs*, *filename*, *extnum*)

drizzlepac.util.**get_expstart**(*header*, *primary_hdr*)
shouldn't this just be defined in the instrument subclass of imageobject?

drizzlepac.util.**get_pool_size**(*usr_config_value*, *num_tasks*)
Determine size of thread/process-pool for parallel processing. This examines the cpu_count to decide and return the right pool size to use. Also take into account the user's wishes via the config object value, if specified. On top of that, don't allow the pool size returned to be any higher than the number of parallel tasks, if specified. Only use what we need (mp.Pool starts pool_size processes, needed or not). If number of tasks is unknown, call this with "num_tasks" set to None. Returns 1 when indicating that parallel processing should not be used. Consolidate all such logic here, not in the caller.

drizzlepac.util.**init_logging**(*logfile='astrodrizzle.log'*, *default=None*, *level=20*)
Set up logger for capturing stdout/stderr messages.

Must be called prior to writing any messages that you want to log.

drizzlepac.util.**isASNTable**(*inputFilelist*)
Return TRUE if inputFilelist is a fits ASN file.

drizzlepac.util.**isCommaList**(*inputFilelist*)
Return True if the input is a comma separated list of names.

drizzlepac.util.**is_blank**(*val*)
Determines whether or not a value is considered 'blank'.

drizzlepac.util.**loadFileList**(*inputFilelist*)
Open up the '@ file' and read in the science and possible ivm filenames from the first two columns.

drizzlepac.util.**parse_colnames**(*colnames*, *coords=None*)
Convert colnames input into list of column numbers.

drizzlepac.util.**printParams**(*paramDictionary*, *all=False*, *log=None*)
Print nicely the parameters from the dictionary.

drizzlepac.util.**print_pkg_versions**(*packages=None*, *svn=False*, *log=None*)

drizzlepac.util.**readCommaList**(*fileList*)
Return a list of the files with the commas removed.

drizzlepac.util.**readcols**(*infile, cols=[0, 1, 2, 3], hms=False*)
Read the columns from an ASCII file as numpy arrays.

Parameters
infile : str

Filename of ASCII file with array data as columns.

cols : list of int

List of 0-indexed column numbers for columns to be turned into numpy arrays (DEFAULT- [0,1,2,3]).

Returns
outarr : list of numpy arrays

Simple list of numpy arrays in the order as specifed in the 'cols' parameter.

drizzlepac.util.**removeFileSafely**(*filename*, *clobber=True*)
Delete the file specified, but only if it exists and clobber is True.

drizzlepac.util.**runmakewcs**(*input*)
Runs 'updatewcs' to recompute the WCS keywords for the input image.

Parameters
input : list of str

A list of file names.

Returns
output : list of str

Returns a list of names of the modified files (For GEIS files returns the translated names).

drizzlepac.util.**updateNEXTENDKw**(*fobj*)
Update NEXTEND keyword in PRIMARY header (if present) to accurately reflect the number of extensions in the MEF file.

drizzlepac.util.**update_input**(*filelist*, *ivmlist=None*, *removed_files=None*)
Removes files flagged to be removed from the input filelist. Removes the corresponding ivm files if present.

drizzlepac.util.**validateUserPars**(*configObj*, *input_dict*)
Compares input parameter names specified by user with those already recognized by the task.

Any parameters provided by the user that does not match a known task parameter will be reported and a ValueError exception will be raised.

drizzlepac.util.**verifyFilePermissions**(*filelist*, *chmod=True*)
Verify that images specified in 'filelist' can be updated.

A message will be printed reporting the names of any images which do not have write-permission, then quit.

drizzlepac.util.**verifyRefimage**(*refimage*)
Verify that the value of refimage specified by the user points to an extension with a proper WCS defined. It starts by making sure an extension gets specified by the user when using a MEF file. The final check comes by looking for a CD matrix in the WCS object itself. If either test fails, it returns a value of False.

drizzlepac.util.**verifyUniqueWcsname**(*fname*, *wcsname*)
Report whether or not the specified WCSNAME already exists in the file

drizzlepac.util.**verifyUpdatewcs**(*fname*)
Verify the existence of WCSNAME in the file. If it is not present, report this to the user and raise an exception. Returns True if WCSNAME was found in all SCI extensions.

10.2 WCS Utilities

These functions read and interpret the WCS information from input images and create the output WCS objects based on STWCS routines.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.wcs_functions.**IdentityMap**(*input*, *output*)

forward(*pixx*, *pixy*)

class drizzlepac.wcs_functions.**LinearMap**(*xsh=0.0*, *ysh=0.0*, *rot=0.0*, *scale=1.0*)

forward(*pixx*, *pixy*)

class drizzlepac.wcs_functions.**WCSMap**(*input*, *output*, *origin=1*)
Sample class to demonstrate how to define a coordinate transformation

backward(*pixx*, *pixy*)
Transform pixx,pixy positions from the output frame back onto their original positions in the input frame.

checkWCS(*obj*, *name*)

forward(*pixx*, *pixy*)
Transform the input pixx,pixy positions in the input frame to pixel positions in the output frame.

This method gets passed to the drizzle algorithm.

get_pix_ratio()
Return the ratio of plate scales between the input and output WCS. This is used to properly distribute the flux in each pixel in 'tdriz'.

rd2xy(*wcs*, *ra*, *dec*)
Transform input sky positions into pixel positions in the WCS provided.

xy2rd(*wcs*, *pixx*, *pixy*)
Transform input pixel positions into sky positions in the WCS provided.

drizzlepac.wcs_functions.**apply_fitlin**(*data*, *P*, *Q*)

drizzlepac.wcs_functions.**build_hstwcs**(*crval1*, *crval2*, *crpix1*, *crpix2*, *naxis1*, *naxis2*, *pscale*, *orientat*)
Create an HSTWCS object for a default instrument without distortion based on user provided parameter values.

drizzlepac.wcs_functions.**build_pixel_transform**(*chip*, *output_wcs*)

drizzlepac.wcs_functions.**calcNewEdges**(*wcs*, *shape*)
This method will compute sky coordinates for all the pixels around the edge of an image AFTER applying the geometry model.

Parameters
wcs : obj

HSTWCS object for image

shape : tuple

numpy shape tuple for size of image

Returns
border : arr

array which contains the new positions for all pixels around the border of the edges in alpha,dec

drizzlepac.wcs_functions.**computeEdgesCenter**(*edges*)

drizzlepac.wcs_functions.**convertWCS**(*inwcs*, *drizwcs*)
Copy WCSObject WCS into Drizzle compatible array.

drizzlepac.wcs_functions.**createWCSObject**(*output*, *default_wcs*, *imageObjectList*)
Converts a PyWCS WCS object into a WCSObject(baseImageObject) instance.

drizzlepac.wcs_functions.**create_CD**(*orient*, *scale*, *cx=None*, *cy=None*)
Create a (un?)distorted CD matrix from the basic inputs

The 'cx' and 'cy' parameters, if given, provide the X and Y coefficients of the distortion as returned by reading the IDCTAB. Only the first 2 elements are used and should correspond to the 'OC[X/Y]10' and 'OC[X/Y]11' terms in that order as read from the expanded SIP headers.

The units of 'scale' should be 'arcseconds/pixel' of the reference pixel. The value of 'orient' should be the absolute orientation on the sky of the reference pixel.

drizzlepac.wcs_functions.**ddtohms**(*xsky*, *ysky*, *verbose=False*, *precision=6*)
Convert sky position(s) from decimal degrees to HMS format.

drizzlepac.wcs_functions.**fitlin**(*imgarr*, *refarr*)
Compute the least-squares fit between two arrays. A Python translation of 'FITLIN' from 'drutil.f' (Drizzle V2.9).

drizzlepac.wcs_functions.**fitlin_clipped**(*xy*, *uv*, *verbose=False*, *mode='rscale'*, *nclip=3*, *reject=3*)
Perform a clipped fit based on the number of iterations and rejection limit (in sigma) specified by the user. This will more closely replicate the results obtained by 'geomap' using 'maxiter' and 'reject' parameters.

drizzlepac.wcs_functions.**fitlin_rscale**(*xy*, *uv*, *verbose=False*)
Performs a linear, orthogonal fit between matched lists of positions 'xy' (input) and 'uv' (output).

Output: (same as for fit_arrays_general)

drizzlepac.wcs_functions.**get_hstwcs**(*filename*, *hdulist*, *extnum*)
Return the HSTWCS object for a given chip.

drizzlepac.wcs_functions.**get_pix_ratio_from_WCS**(*input*, *output*)
[Functional form of .get_pix_ratio() method of WCSMap]

drizzlepac.wcs_functions.**make_outputwcs**(*imageObjectList*, *output*, *configObj=None*, *perfect=False*)
Computes the full output WCS based on the set of input imageObjects provided as input, along with the pre-determined output name from process_input. The user specified output parameters are then used to modify the default WCS to produce the final desired output frame. The input imageObjectList has the outputValues dictionary updated with the information from the computed output WCS. It then returns this WCS as a WCSObject(imageObject) instance.

drizzlepac.wcs_functions.**make_perfect_cd**(*wcs*)
Create a perfect (square, orthogonal, undistorted) CD matrix from the input WCS.

drizzlepac.wcs_functions.**mergeWCS**(*default_wcs*, *user_pars*)
Merges the user specified WCS values given as dictionary derived from the input configObj object with the output PyWCS object computed using distortion.output_wcs().

The user_pars dictionary needs to have the following set of keys:

```
    user_pars = {'ra':None,'dec':None,'scale':None,'rot':None,
                 'outnx':None,'outny':None,'crpix1':None,'crpix2':None}
```

drizzlepac.wcs_functions.**removeAllAltWCS**(*hdulist*, *extlist*)
Removes all alternate WCS solutions from the header

drizzlepac.wcs_functions.**restoreDefaultWCS**(*imageObjectList*, *output_wcs*)
Restore WCS information to default values, and update imageObject accordingly.

drizzlepac.wcs_functions.**updateImageWCS**(*imageObjectList*, *output_wcs*)

drizzlepac.wcs_functions.**updateWCS**(*drizwcs*, *inwcs*)
Copy output WCS array from Drizzle into WCSObject.

drizzlepac.wcs_functions.**update_linCD**(*cdmat, delta_rot=0.0, delta_scale=1.0, cx=[0.0, 1.0], cy=[1.0, 0.0]*)
Modify an existing linear CD matrix with rotation and/or scale changes and return a new CD matrix. If 'cx' and 'cy' are specified, it will return a distorted CD matrix.

Only those terms which are varying need to be specified on input.

drizzlepac.wcs_functions.**wcsfit**(*img_wcs*, *ref_wcs*)
Perform a linear fit between 2 WCS for shift, rotation and scale. Based on the WCSLIN function from 'drutil.f'(Drizzle V2.9) and modified to allow for differences in reference positions assumed by PyDrizzle's distortion model and the coeffs used by 'drizzle'.

Parameters
img : obj

ObsGeometry instance for input image

ref_wcs : obj

Undistorted WCSObject instance for output frame

10.3 Output Image Generation

This module manages the creation of the output image FITS file.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.outputimage.**OutputImage**(*plist*, *input_pars*, *build=True*, *wcs=None*, *single=False*, *blot=False*)
This class manages the creation of the array objects which will be used by Drizzle. The three arrays, SCI/WHT/CTX, will be setup either as extensions in a single multi-extension FITS file, or as separate FITS files.

The object 'plist' must contain at least the following members:

```
    plist['output']   - name of output FITS image (for SCI)
    plist['outnx']    - size of X axis for output array
    plist['outny']    - size of Y axis for output array
```

If 'single=yes', then 'plist' also needs to contain:

```
plist['outsingle']
plist['outsweight']
plist['outscontext']
```

If 'blot=yes', then 'plist' also needs:

```
plist['data']
plist['blotImage']
plist['blotnx'],plist['blotny']
```

If 'build' is set to 'no', then each extension/array must be in separate FITS objects. This would also require:

```
plist['outdata']    - name of output SCI FITS image
plist['outweight']  - name of output WHT FITS image
plist['outcontext'] - name of output CTX FITS image
```

Optionally, the overall exposure time information can be passed as:

```
plist['texptime'] - total exptime for output
plist['expstart'] - start time of combined exposure
plist['expend']   - end time of combined exposure
```

addDrizKeywords(*hdr*, *versions*)
Add drizzle parameter keywords to header.

find_kwupdate_location(*hdr*, *keyword*)
Find the last keyword in the output header that comes before the new keyword in the original, full input headers. This will rely on the original ordering of keywords from the original input files in order to place the updated keyword in the correct location in case the keyword was removed from the output header prior to calling this method.

set_bunit(*bunit*)
Method used to update the value of the bunit attribute.

set_units(*units*)
Method used to record what units were specified by the user for the output product.

writeFITS(*template*, *sciarr*, *whtarr*, *ctxarr=None*, *versions=None*, *overwrite=True*, *blend=True*, *virtual=False*)
Generate PyFITS objects for each output extension using the file given by 'template' for populating headers.

The arrays will have the size specified by 'shape'.

10.4 MultiDrizzle Reference Table

This module supports the interpretation of the `MDRIZTAB` for processing as used in the pipeline. This module supports the interpretation of the `MDRIZTAB` for processing as used in the pipeline.

Authors

Warren Hack, Ivo Busko, Christopher Hanley

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.mdzhandler.**cleanBlank**(*value*)

drizzlepac.mdzhandler.**cleanInt**(*value*)

drizzlepac.mdzhandler.**cleanNaN**(*value*)

drizzlepac.mdzhandler.**findFormat**(*format*)

drizzlepac.mdzhandler.**getMdriztabParameters**(*files*)

Gets entry in MDRIZTAB where task parameters live. This method returns a record array mapping the selected row.

drizzlepac.mdzhandler.**toBoolean**(*flag*)

CHAPTER

ELEVEN

TWEAKBACK

tweakback - propagate the "tweaked" solutions back to the original input files.

Version 0.4.0 - replaced previous algorithm that used fitting of WCS footprints to reconstruct the transformation that was applied to the old drizzled image (to align it with another image) to obtain the new drizzled image WCS with an algorithm that is based on linearization of the exact compound operator that transforms current image coordinates to the "aligned" (to the new drizzled WCS) image coordinates.

Authors
Warren Hack, Mihai Cara

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.tweakback.**determine_extnum**(*drzfile*, *extname='SCI'*)

drizzlepac.tweakback.**determine_orig_wcsname**(*header*, *wnames*, *wkeys*)
Determine the name of the original, unmodified WCS solution

drizzlepac.tweakback.**extract_input_filenames**(*drzfile*)
Generate a list of filenames from a drizzled image's header

drizzlepac.tweakback.**getHelpAsString**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

drizzlepac.tweakback.**help**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.tweakback.**linearize**(*wcsim*, *wcsima*, *wcs_olddrz*, *wcs_newdrz*, *imcrpix*, *hx=1.0*, *hy=1.0*)

drizzlepac.tweakback.**run**(*configobj*)

drizzlepac.tweakback.**tweakback**(*drzfile*, *input=None*, *origwcs=None*, *newname=None*, *wcsname=None*, *extname='SCI'*, *force=False*, *verbose=False*)

Apply WCS solution recorded in drizzled file to distorted input images (_flt.fits files) used to create the drizzled file. This task relies on the original WCS and updated WCS to be recorded in the drizzled image's header as the last 2 alternate WCSs.

Parameters

drzfile : str (Default = '')

filename of undistorted image which contains the new WCS and WCS prior to being updated

newname : str (Default = None)

Value of WCSNAME to be used to label the updated solution in the output (eq., _flt.fits) files. If left blank or None, it will default to using the current WCSNAME value from the input drzfile.

input : str (Default = '')

filenames of distorted images to be updated using new WCS from 'drzfile'. These can be provided either as an @-file, a comma-separated list of filenames or using wildcards.

Note: A blank value will indicate that the task should derive the filenames from the 'drzfile' itself, if possible. The filenames will be derived from the D*DATA keywords written out by astrodrizzle (or MultiDrizzle or drizzle). If they can not be found, the task will quit.

---

origwcs : str (Default = None)

Value of WCSNAME keyword prior to the drzfile image being updated by tweakreg. If left blank or None, it will default to using the second to last WCSNAME* keyword value found in the header.

wcsname : str (Default = None)

Value of WCSNAME for updated solution written out by *tweakreg* as specified by the *wcsname* parameter from *tweakreg*. If this is left blank or *None*, it will default to the current WCSNAME value from the input drzfile.

extname : str (Default = 'SCI')

Name of extension in *input* files to be updated with new WCS

force : bool (Default = False)

This parameters specified whether or not to force an update of the WCS even though WCS already exists with this solution or *wcsname*?

verbose : bool (Default = False)

This parameter specifies whether or not to print out additional messages during processing.

See also:

stwcs.wcsutil.altwcs
Alternate WCS implementation

Notes

The algorithm used by this function is based on linearization of the exact compound operator that converts input image coordinates to the coordinates (in the input image) that would result in alignment with the new drizzled image WCS.

If no input distorted files are specified as input, this task will attempt to generate the list of filenames from the drizzled input file's own header.

Examples

An image named `acswfc_mos2_drz.fits` was created from 4 images using astrodrizzle. This drizzled image was then aligned to another image using tweakreg and the header was updated using the `WCSNAME = TWEAK_DRZ`. The new WCS can then be used to update each of the 4 images that were combined to make up this drizzled image using:

```
>>> from drizzlepac import tweakback
>>> tweakback.tweakback('acswfc_mos2_drz.fits')
```

If the same WCS should be applied to a specific set of images, those images can be updated using:

```
>>> tweakback.tweakback('acswfc_mos2_drz.fits',
...                     input='img_mos2a_flt.fits,img_mos2e_flt.fits')
```

drizzlepac.tweakback.**update_chip_wcs**(*chip_wcs*, *drz_old_wcs*, *drz_new_wcs*, *xrms=None*, *yrms=None*)

drizzlepac.tweakback.**tweakback**(*drzfile*, *input=None*, *origwcs=None*, *newname=None*, *wcsname=None*, *extname='SCI'*, *force=False*, *verbose=False*)

Apply WCS solution recorded in drizzled file to distorted input images (`_flt.fits` files) used to create the drizzled file. This task relies on the original WCS and updated WCS to be recorded in the drizzled image's header as the last 2 alternate WCSs.

Parameters

drzfile : str (Default = '')

filename of undistorted image which contains the new WCS and WCS prior to being updated

newname : str (Default = None)

Value of `WCSNAME` to be used to label the updated solution in the output (eq., `_flt.fits`) files. If left blank or None, it will default to using the current `WCSNAME` value from the input drzfile.

input : str (Default = '')

filenames of distorted images to be updated using new WCS from 'drzfile'. These can be provided either as an `@-file`, a comma-separated list of filenames or using wildcards.

Note: A blank value will indicate that the task should derive the filenames from the 'drzfile' itself, if possible. The filenames will be derived from the `D*DATA` keywords written out by astrodrizzle (or MultiDrizzle or drizzle). If they can not be found, the task will quit.

---

origwcs : str (Default = None)

Value of `WCSNAME` keyword prior to the drzfile image being updated by tweakreg. If left blank or None, it will default to using the second to last `WCSNAME*` keyword value found in the header.

wcsname : str (Default = None)

Value of WCSNAME for updated solution written out by *tweakreg* as specified by the *wcsname* parameter from *tweakreg*. If this is left blank or *None*, it will default to the current `WCSNAME` value from the input drzfile.

extname : str (Default = 'SCI')

Name of extension in *input* files to be updated with new WCS

force : bool (Default = False)

This parameters specified whether or not to force an update of the WCS even though WCS already exists with this solution or *wcsname*?

verbose : bool (Default = False)

This parameter specifies whether or not to print out additional messages during processing.

See also:

stwcs.wcsutil.altwcs
Alternate WCS implementation

Notes

The algorithm used by this function is based on linearization of the exact compound operator that converts input image coordinates to the coordinates (in the input image) that would result in alignment with the new drizzled image WCS.

If no input distorted files are specified as input, this task will attempt to generate the list of filenames from the drizzled input file's own header.

Examples

An image named `acswfc_mos2_drz.fits` was created from 4 images using astrodrizzle. This drizzled image was then aligned to another image using tweakreg and the header was updated using the `WCSNAME` = `TWEAK_DRZ`. The new WCS can then be used to update each of the 4 images that were combined to make up this drizzled image using:

```
>>> from drizzlepac import tweakback
>>> tweakback.tweakback('acswfc_mos2_drz.fits')
```

If the same WCS should be applied to a specific set of images, those images can be updated using:

```
>>> tweakback.tweakback('acswfc_mos2_drz.fits',
...                     input='img_mos2a_flt.fits,img_mos2e_flt.fits')
```

CHAPTER

TWELVE

DRIZZLEPAC RELEASE NOTES

The code for this package gets released through a number of methods: namely, the use of the package for pipeline and archive processing of ACS and WFC3 data, SSB's semi-annual public release of the stsci_python package, and a weekly beta release of the development version. The following notes provide some details on what has been revised for each version.

12.1 DrizzlePac Release Notes

The code for this package gets released through a number of methods: namely,

- the use of the package for pipeline and archive processing of ACS and WFC3 data,
- SSB's semi-annual public release of the stsci_python package, and
- a weekly beta release of the development version as part of the IRAFX download.

12.1.1 Identifying the DrizzlePac Version

The version of DrizzlePac can be identified using:

```
> pyraf
>>> import drizzlepac
>>> drizzlepac.__version__
```

NOTES for each version

The following notes provide some details on what has been revised for each version in reverse chronological order (most recent version at the top of the list).

12.1.2 DrizzlePac(astrodrizzle) v2.1.0

Publicly Released : [TBD]

available under SSBX/IRAFX starting: Nov 2, 2015

This release builds upon the major set of changes implemented in v2.0.0 with a number of bug fixes and API changes. The most significant changes include:

- **[API change]** The 'updatewcs' parameter has been removed from the TEAL interface for both 'astrodrizzle' and 'tweakreg'. Python scripts calling these tasks directly can still explicitly set the 'updatewcs' parameter if necessary.
- **[API change]** Stand-alone interface for the blot routine (ablot.blot()) revised to work seamlessly with astrodrizzle-generated products while being more obvious how to call it correctly. The help file for this task was also heavily revised to document all the input parameters and to provide an example of how to use the task.
- **[API change]** Coordinate transformation task (pixtopix/pixtosky/skytopix) interfaces changed to be more consistent, yet remain backward-compatible for now.
- Both astrodrizzle and tweakreg now return an output CD matrix which has identical cross-terms indicating the same scale and orientation in each axis.

The complete list of updates for v2.1.0 can be found at:

DrizzlePac v2.1.0 Release Notes

This version builds upon the major set of changes implemented in v2.0.0 by not only fixing some bugs, but also cleaning up/changing/revising some APIs and docstrings. The complete list of changes includes:

- The 'updatewcs' parameter was removed from both the 'astrodrizzle' and 'tweakreg' interactive TEAL interfaces. The 'updatewcs' parameter can still be used with the Python interface for both the astrodrizzle.Astrodrizzle() and tweakreg.TweakReg() functions instead of calling the 'stwcs.updatewcs.updatewcs()' function separately before running 'astrodrizzle' or 'tweakreg'.
- The stand-alone interface for the blot routine (ablot.blot()) has been revised to work seamlessly with astrodrizzle-generated products while being more obvious how to call it correctly. The help file for this task was also heavily revised to document all the input parameters and to provide an example of how to use the task.
- Both astrodrizzle and tweakreg now return an output CD matrix which has identical cross-terms indicating the same scale and orientation in each axis. This relies on a revision to the stwcs.distortion.utils.output_wcs() function.
- The user interfaces to all 3 coordinate transformation tasks now use 'coordfile' as the input file of coordinates to transform. The use of 'coords' has been deprecated, but still can be used if needed. However, use of 'coordfile' will always override any input provided simultaneously with 'coords' parameter. Help files have been updated to document this as clearly as possible for users.
- User-provided list of input catalogs no longer needs to be matched exactly with input files. As long as all input images are included in input catalog list in any order, tweakreg will apply the correct catalog to the correct file.
- Tweakreg has been updated to correctly and fully apply source selection criteria for both input source catalogs and reference source catalogs based on fluxmin,fluxmax and nbright for each.
- All use of keyword deletion has been updated in drizzlepac (and fitsblender) to avoid warnings from astropy.

- All 3 coordinate transformation tasks rely on the input of valid WCS information for the calculations. These tasks now warn the user when it could not find a valid WCS and instead defaulted to using a unity WCS, so that the user can understand what input needs to be checked/revised to get the correct results.
- Exclusion/inclusion region files that can be used with 'tweakreg' can now be specified in image coordinates and sky coordinates and will only support files written out using DS9-compatible format.
- The filename for 'final_refimage' in astrodrizzle and 'refimage' in tweakreg can now be specified with OR without an extension, such as '[sci,1]' or '[0]'. If no extension is specified, it will automatically look for the first extension with a valid HSTWCS and use that. This makes the use of this parameter in both place consistent and more general than before.
- The reported fit as written out to a file has been slightly modified to report more appropriate numbers of significant digits for the results.
- Use of astrolib.coords was removed from drizzlepac and replaced by use of astropy functions instead. This eliminated one more obsolete dependency in our software.
- Code was revised to rely entirely on astropy.wcs instead of stand-alone pywcs.
- Code was revised to rely entirely on astropy.io.fits instead of stand-alone pyfits.
- Added *photeq* task to account for inverse sensitivity variations across detector chips and/or epochs.
- WFPC2 data from the archive with DGEOFILE reference files will now need to be processed using 'stwcs.updatewcs' before running them through astrodrizzle or tweakreg. This update converts the obsolete, unsupported DGEOFILE correction for the WFPC2 data into a D2IMFILE specific for each WFPC2 observation, then uses that to convert the WCS based on the new conventions used for ACS and WFC3.

This set of changes represents the last major development effort for DrizzlePac in support of HST. Support of this code will continue throughout the lifetime of HST, but will be limited primarily to bug fixes to keep the code viable as Python libraries used by DrizzlePac continue to develop and evolve with the language.

DrizzlePac v2.0.0 Release Notes

This version encompasses a large number of updates and revisions to the DrizzlePac code, including several parameter name changes and additions. The scope of these changes indicates the level of effort that went into improving the DrizzlePac code to make it easier and more productive for users.

Summary of Revisions

The most significant updates to the DrizzlePac code include:

- The Python code has been updated to work identically (without change) under both Python 2.7 and Python 3.x.
- Implementing sky matching, a new algorithm for matching the sky across a set of images being combined by astrodrizzle
- Updating tweakreg to now align full mosaics where some images may not overlap others in the mosaic

- Added the option to write out single drizzle step images as compressed images (to save disk space for large mosaics, and I/O time for single drizzle step)
- Improved tweakreg residual plots visually while allowing them to be written out automatically when tweakreg gets run in non-interactive mode
- Renamed parameters in tweakreg and imagefind to eliminate name clashes
- Added option to select sources based on sharpness/roundness when tweakreg searches for sources
- Added support for exclusion and inclusion regions arbitrary shape/size when tweakreg searches for sources
- Added a full set of source detection parameters for reference image to support multi-instrument alignment in tweakreg
- Added support for new (simpler, more robust) ACS calibration of time-dependent distortion
- A full 6-parameter general linear fit can now be performed using tweakreg, in addition to shift and rscale
- Cleaned up logic for sky-subtraction: user can now turn off sky-subtraction with skysub=no, and still specify a user-defined sky value as the skyuser keyword. This will reduce(eliminate?) the need to manually set MDRIZSKY=0.

In addition to these major updates/changes, numerous smaller bugs were fixed and other revisions were implemented

- headerlet code now accepts lists of files to be updated
- source sky positions (RA and Dec) now included in match file
- DQ flags can now be taken into account when performing source finding in tweakreg
- all intermediate files generated by astrodrizzle will now be removed when using 'clean'='yes'
- a problem was fixed that caused createMedian to crash where there were no good pixels in one of the images (when they did not overlap)
- interpretation of shiftfile now improved to handle arbitrarily-long filenames, rather than being limited to 24 character filenames
- documentation has been updated, sometimes with a lot more extensive descriptions

This version of DrizzlePac also requires use of the latest release version of astropy primarily for WCS and FITS I/O support.

12.1.3 DrizzlePac(astrodrizzle) v2.0.0

Publicly Released through PyPI: [TBD]

available under SSBX/IRAFX starting: Aug 4, 2014

This major release includes a large number of revisions and bug fixes, including the addition of new tasks and parameters. The full description of all the changes included in this new release:

12.1.4 DrizzlePac(astrodrizzle) v1.1.16

Publicly Released through PyPI: Mar 27, 2014

available under SSBX/IRAFX starting: Mar 13, 2014

- Support for WFPC2 GEIS input images improved to correctly find the associated DQ images.
- Static mask files created for all chips in an image now get deleted when using the 'group' parameter to only drizzle a single chip or subset of chips.
- Fixed problem caused by changes to stsci.tools code so that drizzlepac will reference the correct extensions in input images.

12.1.5 DrizzlePac(astrodrizzle) v1.1.15(30-Dec-2013)

Publicly Released through PyPI: Jan 14, 2014

available under SSBX/IRAFX starting: Jan 6, 2014

- [Bug Fix] Files created or updated by drizzlepac, fitsblender, or STWCS tasks, e.g. tweakreg or apply_headerlet, will now ensure that the NEXTEND keyword value correctly reflects the number of extensions in the FITS file upon completion.

12.1.6 DrizzlePac(astrodrizzle) v1.1.14dev(21-Oct-2013) in IRAFX

Installed in OPUS: Dec 11, 2013

available starting: Oct 28, 2013

- [Bug Fix] DQ arrays in input images now get updated with cosmic-ray masks computed by astrodrizzle when run with the parameter 'in_memory=True'. This restored the cosmic-ray masks detected during pipeline processing.

12.1.7 DrizzlePac(astrodrizzle) v1.1.13dev(11-Oct-2013) in IRAFX

available starting: Oct 21, 2013

- Tweakreg can now be run in 'batch' mode. This allows the user to generate plots and have them saved to disk automatically without stopping processing and requiring any user input.

12.1.8 DrizzlePac(astrodrizzle) v1.1.12dev(05-Sep-2013) in IRAFX

available starting: Sept 9, 2013

This version fixed a couple of bugs in astrodrizzle; namely,

- Logic was updated to support pixfrac = 0.0 without crashing. Ths code will now automatically reset the kernel to 'point' in that case.

- Astrodrizzle now forcibly removes all OPUS WCS keywords from drizzle product headers.
- Default rules for generating drizzle product headers (as used in the archive) were modified to add definitions for 'float_one', 'int_one', 'zero' that generate output values of 1.0, 1, and 0 (zero) respectively for use as keyword values. This allows the LTM* rules to replace 'first' with 'float_one' so that the physical and image coordinates for drizzle products are consistent.

Additionally, changes were made to STWCS for reprocessing use:

- Problems with using apply_headerlet_as_primary() from the STWCS package on WFPC2 data have been corrected in this revision.

12.1.9 DrizzlePac(astrodrizzle) v1.1.11dev(05-Jul-2013) in IRAFX

available starting: July 15, 2013

- AstroDrizzle now can process all STIS data without crashing.

12.1.10 DrizzlePac(astrodrizzle) v1.1.10dev(06-Feb-2013) in IRAFX

available starting: May 6, 2013

- The output drizzle image header no longer contains references to D2IM arrays. This allows tweakreg to work with drizzled images as input where 2-D D2IM corrections were needed.
- Deprecated references to PyFITS .has_key() methods were also removed from the entire package, making it compatible with PyFITS 3.2.x and later.

12.1.11 DrizzlePac(astrodrizzle) v1.1.8dev(06-Feb-2013) in IRAFX

available starting: Feb 11, 2013

- Fixed a bug in astrodrizzle which caused blot to raise an exception when using 'sinc' interpolation.
- Cleaned up the logic for writing out the results from the pixtopix, pixtosky, and skytopix tasks to avoid an Exception when a list of inputs are provided and no output file is specified.
- A new parameter was added to the tweakback task to allow a user to specify the value of WCSNAME when updating the FLT images with a new solution from a DRZ image header.
- Code in tweakback for updating the header with a new WCS will now automatically generate a unique WCSNAME if the there is a WCS solution in the FLT headers with the default or user-defined value of WCSNAME.

12.1.12 DrizzlePac(astrodrizzle) v1.1.7dev(18-Dec-2012) in IRAFX

available starting: Feb 4, 2013

- Updated astrodrizzle to work with input images which do not have WCSNAME defined. This should make it easier to support non-HST input images in the future.
- cleared up confusion between flux parameters in imagefindpars and catalog inputs in tweakreg.
- turned of use of fluxes for trimming input source catalogs when no flux column can be found in input source catalogs

12.1.13 DrizzlePac(astrodrizzle) v1.1.7dev(18-Dec-2012) in IRAFX

available starting: Dec 10, 2012

- Update tweakreg 2d histogram building mode to correctly find the peak when all the inputs match with the same offset (no spurious sources in either source catalog).
- Fixed a bug so that Ctrl-C does not cause an exception when used while tweakreg is running
- revised the source finding logic to ignore sources near the image edge, a change from how daofind works (daofind expands the image with blanks then fits anyway)
- created a new function to apply the nsigma separation criteria to (try to) eliminate duplicate entries for the same source from the source list. It turns out daofind does have problems with reporting some duplicate sources as well. This function does not work perfectly, but works to remove nearly all (if not all) duplicates in most cases.

12.1.14 DrizzlePac(astrodrizzle) v1.1.7dev(8-Jan-2012) in IRAFX

available starting: Jan 14, 2013

- Bug fixed in updatehdr module to allow shiftfiles without RMS columns to work as inputs to manually apply shifts to headers of input images
- Revised astrodrizzle to update WCS of all input images BEFORE checking whether or not they are valid. This ensures that all files provided as input to astrodrizzle in the pipeline have the headers updated with the distortion model and new WCS.
- Images with NGOODPIX=0 now identified for WFC3 and WFPC2 inputs, so they can be ignored during astrodrizzle processing.
- Replaced 2d histogram building code originally written in Python with a C function that run about 4x faster.

12.1.15 DrizzlePac(astrodrizzle) v1.1.6dev(5-Dec-2012) in IRAFX

available starting: Dec 10, 2012

- tweakreg v1.1.0 source finding algorithm now runs many times faster (no algorithmic changes). No changes have been made yet to speed up the 2d histogram source matching code.
- The 'pixtopix' task was updated to make the 'outimage' parameter optional by using the input image as the default. This required no API changes, but the help files were updated

- Very minor update to guard against MDRIZTAB being specified without any explicit path.
- Update astrodrizzle to correctly report the exposure time, exposure start, and exposure end for the single drizzle products, in addition to insuring the final drizzle values remain correct.
- astrodrizzle also includes initial changes to safeguard the C code from getting improperly cast values from the configObj(TEAL) input.

12.1.16 DrizzlePac(astrodrizzle) v1.1.5dev(23-Oct-2012) in IRAFX

available starting: Oct 29, 2012

- Scaling of sky array for WFC3/IR IVM generation now correct
- template mask files for WFPC2 no longer generated so that WFPC2 data can now be processed using num_cores > 1 (parallel processing)
- interpretation of the 'group' parameter fixed to support a single integer, a comma-separated list of integers or a single 'sci,<n>' value. The values correspond to the FITS extension number of the extensions that should be combined. This fix may also speed up the initialization step as more direct use of pyfits was implemented for the interpretation of the 'group' parameter.

12.1.17 DrizzlePac(astrodrizzle) v1.1.1(31-Aug-2012) in HST Archive

available starting: Sept 26, 2012

The HST Archive and operational calibration pipeline started using this version on Sept 26, 2012.

12.1.18 DrizzlePac(astrodrizzle) v1.1.4dev(20-Sep-2012) in IRAFX

available starting: Sept 24, 2012

- Bug fixed to allow use of final_wht_type=IVM for processing WFPC2 data
- Revised Initialization processing to speed it up by using more up-to-date, direct pyfits calls.

12.1.19 DrizzlePac(astrodrizzle) v1.1.3(7-Sep-2012) in IRAFX

available starting: Sept 17, 2012

- Fixed the logic so that crclean images always get created regardless of the value of the 'clean' parameter.

12.1.20 DrizzlePac(astrodrizzle) v1.1.2(5-Sep-2012) in IRAFX

available starting: Sept 10, 2012

- Remove the restriction of only being able to process images which have WCSNAME keyword as imposed by r15631. The removal of this restriction will now allow for processing of non-updated input files with updatewcs=False for cases where no distortion model exists for the data (as required by CADC).
- Added log statements reporting what sky value was actually used in the drizzle and blot steps

12.1.21 DrizzlePac(astrodrizzle) v1.1.1(30-Aug-2012) in IRAFX

available starting: Sept 3, 2012

- Major revision to astrodrizzle allowing the option to process without writing out any intermediate products to disk. The intermediate products remain in memory requiring significantly more memory than usual. This improves the overall processing time by eliminating as much disk activity as possible as long as the OS does not start disk swapping due to lack of RAM.
- revised to turn off 'updatewcs' when coeffs=False(no) so that exposures with filter combinations not found in the IDCTAB will not cause an error

12.1.22 DrizzlePac(astrodrizzle) v1.0.7(21-Aug-2012) in IRAFX

available starting: Aug 27, 2012

- Fixes problems with missing single_sci images.
- Static mask step revised to skip updates to static mask if all pixel data falls within a single histogram bin. This avoids problems with masking out entire images, which happens if low S/N SBC data is processed with static_mask=yes.

12.1.23 DrizzlePac(astrodrizzle) v1.0.6(14-Aug-2012) in IRAFX

available starting: Aug 20, 2012

Use of IVM for final_wht now correct, as previous code used wrong inputs when IVM weighting was automatically generated by astrodrizzle.

12.1.24 DrizzlePac(astrodrizzle) v1.0.5(8-Aug-2012) in IRAFX

available starting: Aug 13, 2012

- Completely removed the use of the TIME arrays for weighting IR drizzle products so that the photometry for saturated sources in drizzled products now comes out correct.
- Corrected a problem with astrodrizzle which affected processing of WFPC2 data where CRPIX2 was not found when creating the output single sci image.

12.1.25 stsci_python v2.13 [Includes astrodrizzle v1.0.2(13-July-2012)]

available starting: Aug 3, 2012

The complete version of stsci_python can be downloaded from our download page

- stsci_python v2.13 Release Notes
- Old stsci_python release notes

12.1.26 DrizzlePac(astrodrizzle) v1.0.1(20-June-2012)

Used in archive/pipeline starting: July 10, 2012

Pipeline and archive started processing ACS data with this version.

12.1.27 DrizzlePac(astrodrizzle) v1.0.0(25-May-2012)

Used in archive/pipeline starting: June 6, 2012

Pipeline and archive first started using astrodrizzle by processing WFC3 images.

CHAPTER

THIRTEEN

IMAGE REGISTRATION TASKS

Documentation for the replacement task for IRAF's *tweakshifts*, currently named *tweakreg*, has been added to this package. These new modules describe how to run the new TEAL-enabled task, as well as use the classes in the task to generate catalogs interactively for any chip and work with that catalog. The current implementation of this code relies on a very basic source finding algorithm loosely patterned after the DAOFIND algorithm and does not provide all the same features or outputs found in DAOFIND. The fitting algorithm also reproduces the fitting performed by IRAF's *geomap* in a limited fashion; primarily, it only performs fits equivalent to *geomap*'s 'shift' and 'rscale' solutions. These algorithms will be upgraded as soon as replacements are available.

13.1 TWEAKREG: Alignment of Images

Combining images using astrodrizzle requires that the WCS information in the headers of each input image align to within sub-pixel accuracy. The *tweakreg* task allows the user to align sets of images to each other and/or to and external astrometric reference frame or image. `TweakReg` - A replacement for IRAF-based *tweakshifts*

Authors
Warren Hack, Mihai Cara

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

`drizzlepac.tweakreg.`**`TweakReg`**(*files=None*, *editpars=False*, *configobj=None*, *imagefindcfg=None*, *refimagefindcfg=None*, ***input_dict*)

Tweakreg provides an automated interface for computing residual shifts between input exposures being combined using *AstroDrizzle*. The offsets computed by Tweakreg correspond to pointing differences after applying the WCS information from the input image's headers. Such errors would, for example, be due to errors in guide-star positions when combining observations from different observing visits or from slight offsets introduced upon re-acquiring the guide stars in a slightly different position.

Parameters

file : str or list of str (Default = `'*flt.fits'`)

Input files (passed in from *files* parameter) This paramater can be provided in any of several forms:

- filename of a single image

- filename of an association (ASN)table
- wild-card specification for files in directory (using `\*`, `?` etc.)
- comma-separated list of filenames
- `@file` filelist containing list of desired input filenames (and optional inverse variance map filenames)

The `@file` filelist needs to be provided as an ASCII text file containing a list of filenames for all input images with one filename on each line of the file. If inverse variance maps have also been created by the user and are to be used (by specifying `IVM` to the parameter *final_wht_type* described further below), then these are simply provided as a second column in the filelist, with each `IVM` filename listed on the same line as a second entry, after its corresponding exposure filename.

editpars : bool (Default = False)

A parameter that allows user to edit input parameters by hand in the GUI. True to use the GUI to edit parameters.

configObj : configObject (Default = None)

An instance of *configObject* which overrides default parameter settings.

imagefindcfg : dict, configObject (Default = None)

An instance of *dict* or *configObject* which overrides default source finding (for input images) parameter settings. See help for `imagefindpars` PSET for a list of available parameters. **Only** the parameters that are different from default values **need** to be specified here.

refimagefindcfg : dict, configObject (Default = None)

An instance of *dict* or *configObject* which overrides default source finding (for input reference image) parameter settings. See help for `refimagefindpars` PSET for a list of available parameters. **Only** the parameters that are different from default values **need** to be specified here.

inputDict : dict, optional

An optional list of parameters specified by the user, which can also be used to override the defaults.

Note: This list of parameters **can** include the `updatewcs` parameter, even though this parameter no longer can be set through the TEAL GUI.

Note: This list of parameters **can** contain parameters specific to the *TweakReg* task itself described here in the "Other Parameters" section and **may not** contain parameters from the `refimagefindpars` PSET.

Note: For compatibility purpose with previous *TweakReg* versions, *inputDict* may contain parameters from the the `imagefindpars` PSET. However, if *imagefindcfg* is not *None*, then `imagefindpars` parameters specified through *inputDict* may not duplicate parameters specified through *imagefindcfg*.

Other Parameters

refimage : str (Default = '')

Filename of reference image. Sources derived from this image will be used as the reference for matching with sources from all input images unless a separate catalog is provided through the *refcat* parameter.

expand_refcat : bool (Default = False)

Specifies whether to add new sources from just matched images to the reference catalog to allow next image to be matched against an expanded reference catalog.

enforce_user_order : bool (Default = True)

Specifies whether images should be aligned in the order specified in the *file* input parameter or *TweakReg* should optimize the order of alignment by intersection area of the images. Default value (*True*) will align images in the user specified order, except when some images cannot be aligned in which case *TweakReg* will optimize the image alignment order. Alignment order optimization is available *only* when *expand_refcat* = *True*.

exclusions: string (Default = '')

This parameter allows the user to specify a set of files which contain regions in the image to ignore (or to include) when finding sources. This file MUST have 1 line for each input image with the name of the input file in the first column. Subsequent columns would be used to specify an inclusion and/or exclusion file for each chip in `'SCI,<n>'` index order. If a chip does not require an exclusion file, the string *None* or `INDEF` can be used as a placeholder for that chip. Each exclusion file can be either a mask provided as a simple FITS file or a region file in DS9-format.

When a mask file is provided, *TweakReg* will look for the first image-like extension with image data of the same dimensions as the input image. Zeros in the mask will be interpreted as "bad" (excluded from search) pixels while non-zero pixels will be interpreted as "good" pixels. It is recommended that mask files be FITS files without extensions and mask data (preferably of integer type) reside in the primary HDU.

If a region file is provided then it should conform to the 'region' file format generated by DS9. The region files can contain both regular ("include") regions as well as "exclude" regions. Regular ("include") regions indicate the regions of the image that should be searched for sources while "exclude"

regions indicate parts of the image that should not be used for source detection. The "ruler", "compass", and "projection" regions are not supported (ignored). When **all regions** in a region file are "exclude" regions, then it will be assumed that the entire image is "good" before the exclude regions are processed. In other words, an "include" region corresponding to the entire image will be *prepended* to the list of exclude regions.

Note: Regions in a region file are processed in the order they appear in the region file. Thus, when region files contain *both* "include" and "exclude" regions, the order in which these regions appear may affect the results.

Warning: *TweakReg* relies on `pyregion` package for work with region files. At the time of writing, `pyregion` uses a different algorithm from DS9 for converting regions from sky coordinates to image coordinate (this conversion is performed before regions are converted to masks). For these reasons, regions provided in sky coordinates may not produce the expected (from DS9) results. While in most instances these discrepancies should be tolerable, it is important to keep this in mind.
During testing it was observed that conversion to image coordinates is most accurate for polygonal regions and less accurate for other regions Therefore, if one must provide regions in sky coordinates, it is types. recommended to use polygonal and circular regions and to avoid elliptical and rectangular regions as their conversion to image coordinates is less accurate. One may use `mapreg` task in the `drizzlepac` package to convert region files from sky coordinates to image coordinates. This will allow one to see the actual regions that will be used by source finding routine in *TweakReg*.

updatewcs : bool (Default = No)

NOT available through TEAL GUI interface. This parameter can only be set through the Python interface to Tweakreg by passing it in as part of the input_dict in order to insure that running `updatewcs` **does not overwrite** a previously determined solution written out to the input file headers.

writecat : bool (Default = Yes)

Specify whether or not to write out the source catalogs generated for each input image by the built-in source extraction algorithm.

clean : bool (Default = No)

Specify whether or not to remove the temporary files created by *tweakreg*, including any catalog files generated for the shift determination.

interactive : bool (Default = Yes)

This switch controls whether the program stops and waits for the user to examine any generated plots before continuing on to the next image. If turned off, plots will still be displayed, but they will also be saved to disk automati-

cally as a PNG image with an autogenerated name without requiring any user input.

verbose : bool (Default = No)

Specify whether or not to print extra messages during processing.

runfile : string (Default = 'tweakreg.log')

Specify the filename of the processing log.

UPDATE HEADER

updatehdr : bool (Default = No)

Specify whether or not to update the headers of each input image directly with the shifts that were determined. This will allow the input images to be combined by *AstroDrizzle* without having to provide the shiftfile as well.

wcsname : str (Default = 'TWEAK')

Name of updated WCS.

reusename : bool (Default = False)

Allows overwriting of an existing WCS with the same name as specified by *wcsname* parameter.

HEADERLET CREATION

headerlet: bool (Default = No)

Specify whether or not to generate a headerlet from the images at the end of the task? If turned on, this will create a headerlet from the images regardless of the value of the *updatehdr* parameter.

attach: bool (Default = Yes)

If creating a headerlet, choose whether or not to attach the new headerlet to the input image as a new extension.

hdrfile: string (Default = '')

Filename to use for writing out headerlet to a separate file. If the name does not contain `.fits`, it will create a filename from the rootname of the input image, the value of this string, and it will end in `'_hlet.fits'`. For example, if only `'hdrlet1'` is given, the full filename created will be `'j99da1f2q_hdrlet1_hlet.fits'` when creating a headerlet for image `'j99da1f2q_flt.fits'`.

clobber: bool (Default = No)

If a headerlet with 'hdrfile' already exists on disk, specify whether or not to overwrite that previous file.

hdrname: string (Default = '')

Unique name to give to headerlet solution. This name will be used to identify this specific WCS alignment solution contained in the headerlet.

author: string, optional (Default = '')

Name of the creator of the headerlet.

descrip: string, optional (Default = '')

Short (1-line) description to be included in headerlet as `DESCRIP` keyword. This can be used to provide a quick look description of the WCS alignment contained in the headerlet.

catalog: string, optional (Default = '')

Name of reference catalog used as the basis for the image alignment.

history: string, optional (Default = '')

Filename of a file containing detailed information regarding the history of the WCS solution contained in the headerlet. This can include information on the catalog used for the alignment, or notes on processing that went into finalizing the WCS alignment stored in this headerlet. This information will be reformatted as 70-character wide FITS HISTORY keyword section.

OPTIONAL SHIFTFILE OUTPUT

shiftfile : bool (Default = No)

Create output shiftfile?

outshifts : str (Default = 'shifts.txt')

The name for the output shift file created by *tweakreg*. This shiftfile will be formatted for use as direct input to *AstroDrizzle*.

outwcs : str (Default = 'shifts_wcs.fits')

Filename to be given to the OUTPUT reference WCS file created by *tweakreg*. This reference WCS defines the WCS from which the shifts get measured, and will be used by *AstroDrizzle* to interpret those shifts. This reference WCS file will be a FITS file that only contains the WCS keywords in a Primary header with no image data itself. The values will be derived from the FIRST input image specified.

COORDINATE FILE DESCRIPTION

catfile : str (Default = '')

Name of file that contains a list of input images and associated catalog files generated by the user. Each line of this file will contain the name of an input image in the first column. The remaining columns will provide the names of the source catalogs for each chip in order of the science extension numbers ((SCI,1), (SCI,2), ...).

A sample catfile, with one line per image would look like:

```
image1_flt.fts  cat1_sci1.coo  cat1_sci2.coo
image2_flt.fts  cat2_sci1.coo  cat2_sci2.coo
```

xcol : int (Default = 1)

Column number of X position from the user-generated catalog files specified in the catfile.

ycol : int (Default = 2)

Column number of Y position from the user-generated catalog files specified in the catfile.

fluxcol : int (Default = None)

Column number for the flux values from the user-generated catalog files specified in the catfile. These values will only be used if a flux limit has been specified by the user using the *maxflux* or *minflux* parameters.

maxflux : float (Default = None)

Limiting flux value for selecting valid objects in the input image's catalog. If specified, this flux will serve as the upper limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to *None*, all objects with fluxes brighter than the minimum specified in *minflux* will be used. If both values are set to *None*, all objects will be used.

minflux : float (Default = None)

Limiting flux value for selecting valid objects in the input image's catalog. If specified, this flux value will serve as the lower limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to *None*, all objects fainter than the limit specified by *maxflux* will be used. If both values are set to *None*, all objects will be used.

fluxunits : str {'counts', 'cps', 'mag'} (Default = 'counts')

This allows the task to correctly interpret the flux limits specified by *maxflux* and *minflux* when sorting the object list for trimming of fainter objects.

xyunits : str {'pixels', 'degrees'} (Default = 'pixels')

Specifies whether the positions in this catalog are already sky pixel positions, or whether they need to be transformed to the sky.

nbright : int (Default = None)

The number of brightest objects to keep after sorting the full object list. If nbright is set equal to *None*, all objects will be used.

REFERENCE CATALOG DESCRIPTION

refcat : str (Default = '')

Name of the external reference catalog file to be used in place of the catalog extracted from one of the input images. When *refimage* is not specified, reference WCS to be used with reference catalog will be derived from input images.

refxcol : int (Default = 1)

Column number of RA in the external catalog file specified by the refcat.

refycol : int (Default = 2)

Column number of Dec in the external catalog file specified by the refcat.

refxyunits : str {'pixels','degrees'} (Default = 'degrees')

Units of sky positions.

rfluxcol : int (Default = None)

Column number of flux/magnitude values in the external catalog file specified by the refcat.

rmaxflux : float (Default = None)

Limiting flux value used to select valid objects in the external catalog. If specified, the flux value will serve as the upper limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to *None*, all objects with fluxes brighter than the minimum specified in *rminflux* will be used. If both values are set to *None*, all objects will be used.

rminflux : float (Default = None)

Limiting flux value used to select valid objects in the external catalog. If specified, the flux will serve as the lower limit of a range for selecting objects to be used in matching with objects identified in the reference image. If the value is set to *None*, all objects fainter than the limit specified by *rmaxflux* will be used. If both values are set to *None*, all objects will be used.

rfluxunits : {'counts', 'cps', 'mag'} (Default = 'mag')

This allows the task to correctly interpret the flux limits specified by *rmaxflux* and *rminflux* when sorting the object list for trimming of fainter objects.

refnbright : int (Default = None)

Number of brightest objects to keep after sorting the full object list. If refnbright is set to *None*, all objects will be used. Used in conjunction with refcat.

OBJECT MATCHING PARAMETERS

minobj : int (Default = 15)

Minimum number of identified objects from each input image to use in matching objects from other images.

searchrad : float (Default = 1.0)

The search radius for a match.

searchunits : str (Default = 'arcseconds')

Units for search radius.

use2dhist : bool (Default = Yes)

Use 2d histogram to find initial offset?

see2dplot : bool (Default = Yes)

See 2d histogram for initial offset?

tolerance : float (Default = 1.0)

The matching tolerance in pixels after applying an initial solution derived from the 'triangles' algorithm. This parameter gets passed directly to *xyxymatch* for use in matching the object lists from each image with the reference image's object list.

separation : float (Default = 0.0)

The minimum separation for objects in the input and reference coordinate lists. Objects closer together than 'separation' pixels are removed from the input and reference coordinate lists prior to matching. This parameter gets passed directly to *xyxymatch* for use in matching the object lists from each image with the reference image's object list.

xoffset : float (Default = 0.0)

Initial estimate for the offset in X between the images and the reference frame. This offset will be used for all input images provided. If the parameter value is set to *None*, no offset will be assumed in matching sources in *xyxymatch*.

yoffset : float (Default = 0.0)

Initial estimate for the offset in Y between the images and the reference frame. This offset will be used for all input images provided.If the parameter value is set to None, no offset will be assumed in matching sources in *xyxymatch*.

CATALOG FITTING PARAMETERS

fitgeometry : str {'shift', 'rscale', 'general'} (Default = 'rscale')

The fitting geometry to be used in fitting the matched object lists. This parameter is used in fitting the offsets, rotations and/or scale changes from the matched object lists. The 'general' fit geometry allows for independent scale and rotation for each axis.

residplot : str {'No plot', 'vector', 'residuals', 'both'} (Default = 'both')

Plot residuals from fit? If 'both' is selected, the 'vector' and 'residuals' plots will be displayed in separate plotting windows at the same time.

nclip : int (Default = 3)

Number of clipping iterations in fit.

sigma : float (Default = 3.0)

Clipping limit in sigma units.

ADVANCED PARAMETERS AVAILABLE FROM COMMAND LINE

updatewcs : bool (Default = No)

This parameter specifies whether the WCS keywords are to be updated by running updatewcs on the input data, or left alone. The update performed by updatewcs not only recomputes the WCS based on the currently used `IDCTAB`, but also populates the header with the `SIP` coefficients. For `ACS/WFC` images, the time-dependence correction will also be applied to the `WCS` and `SIP` keywords. This parameter should be set to 'No' (*False*) when the WCS keywords have been carefully set by some other method, and need to be passed through to drizzle 'as is', otherwise those updates will be over-written by this update.

Note: This parameter was preserved in the API for compatibility purposes with existing user processing pipe-lines. However, it has been removed from the TEAL interface because it is easy to have it set to 'yes' (especially between consecutive runs of *AstroDrizzle*) with potentially disastrous effects on input image WCS (for example it could wipe-out previously aligned WCS).

See also:

`astrodrizzle`

Notes

Tweakreg supports the use of calibrated, distorted images (such as FLT images for ACS and WFC3, or `_c0m.fits` images for WFPC2) as input images. All coordinates for sources derived from these images (either by this task or as provided by the user directly) will be corrected for distortion using the distortion model information specified in each image's header. This eliminates the need to run 'astrodrizzle' on the input images prior to running *tweakreg*.

Note: All calibrated input images must have been updated using `updatewcs` from the *STWCS* package, to include the full distortion model in the header. Alternatively, one can set `updatewcs` parameter to *True* when running either *TweakReg* or *AstroDrizzle* from command line (Python interpreter) **the first time** on such images.

This task will use catalogs, and catalog-matching, based on the *xyxymatch* algorithm to determine the offset between the input images. The primary mode of operation will be to extract a catalog of source positions from each input image using either a 'DAOFIND-like' algorithm or `SExtractor` (if the user has `SExtractor` installed). Alternatively, the user can provide their catalogs of source positions derived from **each input chip**.

The reference frame will be defined either by:

- the image with the largest overlap with another input image AND with the largest total overlap with the rest of the input images,
- a catalog derived from a reference image specified by the user, or

•a catalog of undistorted sky positions (RA/Dec) and fluxes provided by the user.

For a given observation, the distortion model is applied to all distorted input positions, and the sources from each chip are then combined into a single catalog of undistorted positions.

The undistorted positions for each observation then get passed to *xyxymatch* for matching to objects from the reference catalog.

The source lists from each image will generally include cosmic-rays as detected sources, which can at times significantly confuse object identification between images. Observations that include long exposures often have more cosmic-ray events than source objects. As such, isolating the cosmic-ray events in those cases would significantly improve the efficiency of common source identification between images. One such method for trimming potential false detections from each source list would be to set a flux limit to exclude detections below that limit. As the fluxes reported in the default source object lists are provided as magnitude values, setting the *maxflux* or *minflux* parameter value to a magnitude- based limit, and then setting the *ascend* parameter to *True*, will allow for the creations of catalogs trimmed of all sources fainter than the provided limit. The trimmed source list can then be used in matching sources between images and in establishing the final fitting for the shifts.

A fit can then be performed on the matched set of positions between the input and the reference to produce the 'shiftfile'. If the user is confident that the solution will be correct, the header of each input image can be updated directly with the fit derived for that image. Otherwise, the 'shiftfile' can be passed to AstroDrizzle for aligning the images.

Note: Because of the nature of the used algorithm it may be necessary to run this task multiple time until new shifts, rotations, and/or scales are small enough for the required precision.

New sources (that are not in the reference catalog) from the matched images are added to the reference catalog in order to allow next image to be matched to a larger reference catalog. This allows alignment of images that do not overlap directly with the reference image and/or catalog and it is particularly useful in image registration of large mosaics. Addition of new sources to the reference catalog can be turned off by setting *expand_refcat* to *False* when using an external reference catalog. When an external catalog is not provided (*refcat*='') *or when using an external reference catalog with 'expand_refcat* set to *True* (assuming *writecat* = *True* and *clean* = *False*), the list of all sources in the expanded reference catalog is saved in a catalog file named `cumulative_sky_refcat_###.coo` where ### is the base file name derived from either the external catalog (if provided) or the name of the image used as the reference image.

When *enforce_user_order* is *False*, image catalogs are matched to the reference catalog in order of decreasing overlap area with the reference catalog, otherwise user order of files specified in the *file* parameter is used.

Format of Exclusion Catalog

The format for the exclusions catalog requires 1 line in the file for every input image, regardless of whether or not that image has any defined exclusion regions. A sample file would look like:

```
j99da1emq_flt.fits
j99da1f2q_flt.fits test_exclusion.reg
```

This file specifies no exclusion files for the first image, and only an regions file for SCI,1 of the second image. NOTE: The first file can be dropped completely from the exclusion catalog file.

In the above example, should an exclusion regions file only be needed for the second chip in the second image, the file would need to look like:

```
j99da1emq_flt.fits
j99da1f2q_flt.fits None test_sci2_exclusion.reg
```

The value *None* could also be replaced by `INDEF` if desired, but either string needs to be present to signify no regions file for that chip while the code continues parsing the line to find a file for the second chip.

Format of Region Files

The format of the exclusions catalogs referenced in the 'exclusions' file defaults to the format written out by DS9 using the 'DS9/Funtools' region file format. A sample file with circle() regions will look like:

```
# Region file format: DS9 version 4.1
# Filename: j99da1f2q_flt.fits[SCI]
global color=green dashlist=8 3 width=1 font="helvetica 10 normal roman"
select=1 highlite=1 dash=0 fixed=0 edit=1 move=1 delete=1 include=1 source=1
image
circle(3170,198,20)
ellipse(3269,428,30,10,45) # a rotated ellipse
box(3241.1146,219.78132,20,20,15) # a rotated box
circle(200,200,50)  # outer circle
-circle(200,200,30) # inner circle
```

This region file will be interpreted as "find all sources in the image that **are inside** the four regions above but **not inside** the region -circle(200,200,30)". Effectively we will instruct *TweakReg* to find all the sources *inside* the following regions:

```
circle(3170,198,20)
ellipse(3269,428,30,10,45) # a rotated ellipse
box(3241.1146,219.78132,20,20,15) # a rotated box
annulus(200,200,30,50)  # outer circle(r=50) - inner circle(r=30)
```

Examples

The tweakreg task can be run from either the TEAL GUI or from the command-line using PyRAF or Python. These examples illustrate the various syntax options available.

Example 1: Align a set of calibrated (`_flt.fits`) images using `IMAGEFIND`, a built-in source finding algorithm based on `DAOPHOT`. Auto-detect the sky sigma value and select sources > 200 sigma. (Auto-sigma is computed from the first input exposure as: `1.5*imstat(image,nclip=3,fields='stddev').`) Set the convolution kernel width to `~2x` the value of the PSF FWHM. Save the residual offsets (`dx`, `dy`, `rot`, `scale`, `xfit_rms`, `yfit_rms`) to a text file.

1.Run the task from PyRAF using the TEAL GUI:

```
>>> import drizzlepac
>>> epar tweakreg
```

2. Run the task from PyRAF using the command line while individually specifying source finding parameters for the reference image and input images:

```
>>> import drizzlepac
>>> from drizzlepac import tweakreg
>>> tweakreg.TweakReg('*flt.fits',
...       imagefindcfg={'threshold' : 200, 'conv_width' : 3.5},
...       refimagefindcfg={'threshold' : 400, 'conv_width' : 2.5},
...       updatehdr=False, shiftfile=True, outshifts='shift.txt')
```

or, using *dict* constructor,

```
>>> import drizzlepac
>>> from drizzlepac import tweakreg
>>> tweakreg.TweakReg('*flt.fits',
...       imagefindcfg=dict(threshold=200, conv_width=3.5),
...       refimagefindcfg=dict(threshold=400, conv_width=2.5),
...       updatehdr=False, shiftfile=True, outshifts='shift.txt')
```

Or, run the same task from the PyRAF command line, but specify all parameters in a config file named "myparam.cfg":

```
>>> tweakreg.TweakReg('*flt.fits', configobj='myparam.cfg')
```

Alternately, edit the imagefind parameters in a TEAL GUI window prior to running the task:

```
>>> tweakreg.edit_imagefindpars()
```

3. Help can be accessed via the "Help" pulldown menu in the TEAL GUI. It can also be accessed from the PyRAF command-line and saved to a text file:

```
>>> from drizzlepac import tweakreg
>>> tweakreg.help()
```

or

```
>>> tweakreg.help(file='help.txt')
>>> page help.txt
```

drizzlepac.tweakreg.**help**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

13.2 Refmagefindpars: Source finding parameters for the reference image

This interface provides a mechanism for setting the parameters used by the built-in source finding algorithm based on the published algorithms used by DAOFIND. These settings apply only to source finding in the reference images. These parameters control the operation of the algorithm that extracts sources from the reference image (if specified) as called by *TWEAKREG: Alignment of Images*. The algorithm implemented follows the concepts defined by `DAOFIND` (without actually using any `DAOFIND` code).

Attributes

computesig: bool (Default = True)
This parameter controls whether or not to automatically compute a sigma value to be used for object identification. If set to `True`, then the value computed will override any user input for the parameter `skysigma`. The automatic sigma value gets computed from each input exposure as:

$$\sigma = \sqrt{2\,|mode|}$$

This single value will then be used for object identification for all input exposures.

skysigma: float (Default = 0.0)
The standard deviation of the sky pixels. This value will only be used if computesig is `False`.

conv_width: float (Default = 3.5)
The convolution kernel width in pixels. Recommended values (~2x the PSF FWHM): ACS/WFC & WFC3/UVIS ~3.5 pix and WFC3/IR ~2.5 pix.

peakmin: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is greater than this value.

peakmax: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is less than this value.

threshold: float (Default = 4.0)
The object detection threshold above the local background in units of sigma.

nsigma: float (Default = 1.5)
The semi-major axis of the Gaussian convolution kernel used to compute the density enhancement and mean density images in Gaussian sigma.

ratio: float (Default = 1.0)
The ratio of the sigma of the Gaussian convolution kernel along the minor axis direction to the sigma along the major axis direction. For a circularly-symmetric kernel use ratio = 1.0.

theta: float (Default = 0.0)
The position angle (degrees) of the major axis of the Gaussian convolution kernel. Theta is measured counter-clockwise from the x axis.

fluxmin: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is greater than this value.

fluxmax: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is less than this value.

dqbits: int, str, None, optional (Default = None)
Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for source finding. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for source finding, then *dqbits* should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting *dqbits* to 12.

Setting *dqbits* to 0 will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for source finding.

The default value of *None* will turn off the use of image's DQ array for source finding.

In order to reverse the meaning of the *dqbits* parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for source finding and to consider as "good" all other pixels (regardless of their DQ flag), set *dqbits* to `~4+8`, or `~4,8`. To obtain the same effect with an *int* input value (except for 0), enter -(4+8+1)=-9. Following this convention, a *dqbits* string value of `'~0'` would be equivalent to setting `dqbits=None`.

use_sharp_round: bool (Default = False)
This parameter controls whether or not to enable selection of sources based on their sharpness and roundness statistics.

sharplo: float (Default = 0.2)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness above the `sharplo` value will be selected.

sharphi: float (Default = 1.0)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due

to bad pixels rather than to astronomical objects. Only sources with sharpness below the `sharphi` value will be selected.

roundlo: float (Default = -1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness above the `roundlo` value will be selected.

roundhi: float (Default = 1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness below the `roundhi` value will be selected.

`drizzlepac.refimagefindpars.`**`getHelpAsString`**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

`drizzlepac.refimagefindpars.`**`help`**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

13.3 Imagefindpars: Source finding parameters

This interface provides a mechanism for setting the parameters used by the built-in source finding algorithm based on the published algorithms used by DAOFIND. These parameters control the operation of the algorithm that extracts sources from the image as called by *TWEAKREG: Alignment of Images*. The algorithm implemented follows the concepts defined by `DAOFIND` (without actually using any `DAOFIND` code).

Attributes

computesig: bool (Default = True)
This parameter controls whether or not to automatically compute a sigma value to be used for object identification. If set to `True`, then the value computed will override any user input for the parameter `skysigma`. The automatic sigma value gets computed from each input exposure as:

$$\sigma = \sqrt{2\,|mode|}$$

This single value will then be used for object identification for all input exposures.

skysigma: float (Default = 0.0)
The standard deviation of the sky pixels. This value will only be used if computesig is `False`.

conv_width: float (Default = 3.5)
The convolution kernel width in pixels. Recommended values (~2x the PSF FWHM): ACS/WFC & WFC3/UVIS ~3.5 pix and WFC3/IR ~2.5 pix.

peakmin: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is greater than this value.

peakmax: float, None (Default = None)
This parameter allows the user to select only those sources whose peak value (in the units of the input image) is less than this value.

threshold: float (Default = 4.0)
The object detection threshold above the local background in units of sigma.

nsigma: float (Default = 1.5)
The semi-major axis of the Gaussian convolution kernel used to compute the density enhancement and mean density images in Gaussian sigma.

ratio: float (Default = 1.0)
The ratio of the sigma of the Gaussian convolution kernel along the minor axis direction to the sigma along the major axis direction. For a circularly-symmetric kernel use ratio = 1.0.

theta: float (Default = 0.0)
The position angle (degrees) of the major axis of the Gaussian convolution kernel. Theta is measured counter-clockwise from the x axis.

fluxmin: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is greater than this value.

fluxmax: float, None (Default = None)
This parameter allows the user to select only those sources whose total flux (in the units of the input image) is less than this value.

dqbits: int, str, None, optional (Default = None)
Integer sum of all the DQ bit values from the input image's DQ array that should be considered "good" when building masks for source finding. For example, if pixels in the DQ array can be combinations of 1, 2, 4, and 8 flags and one wants to consider DQ "defects" having flags 2 and 4 as being acceptable for source finding, then *dqbits* should be set to 2+4=6. Then a DQ pixel having values 2,4, or 6 will be considered a good pixel, while a DQ pixel with a value, e.g., 1+2=3, 4+8=12, etc. will be flagged as a "bad" pixel.

Alternatively, one can enter a comma- or '+'-separated list of integer bit flags that should be added to obtain the final "good" bits. For example, both `4,8` and `4+8` are equivalent to setting *dqbits* to 12.

Setting *dqbits* to 0 will make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels will not be used for source finding.

The default value of *None* will turn off the use of image's DQ array for source finding.

In order to reverse the meaning of the *dqbits* parameter from indicating values of the "good" DQ flags to indicating the "bad" DQ flags, prepend '~' to the string value. For example, in order not to use pixels with DQ flags 4 and 8 for source finding and to consider as "good" all other pixels (regardless of their DQ flag), set *dqbits* to `~4+8`, or `~4,8`. To obtain the same effect with an *int* input value (except for 0), enter -(4+8+1)=-9. Following this convention, a *dqbits* string value of `'~0'` would be equivalent to setting `dqbits=None`.

use_sharp_round: bool (Default = False)
This parameter controls whether or not to enable selection of sources based on their sharpness and roundness statistics.

sharplo: float (Default = 0.2)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness above the `sharplo` value will be selected.

sharphi: float (Default = 1.0)
`sharplo` and `sharphi` are designed to eliminate brightness maxima that are due to bad pixels rather than to astronomical objects. Only sources with sharpness below the `sharphi` value will be selected.

roundlo: float (Default = -1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness above the `roundlo` value will be selected.

roundhi: float (Default = 1.0)
`roundlo` and `roundhi` are designed to eliminate brightness maxima that are due to bad rows or columns, rather than to astronomical objects. Only sources with roundness below the `roundhi` value will be selected.

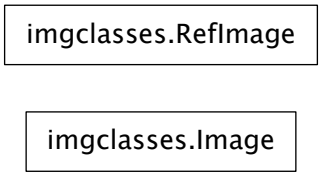
`drizzlepac.imagefindpars.`**`getHelpAsString`**(*docstring=False*, *show_ver=True*)
return useful help from a file in the script directory called __taskname__.help

`drizzlepac.imagefindpars.`**`help`**(*file=None*)
Print out syntax help for running astrodrizzle

Parameters
file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

13.4 Image Class



Classes to keep track of all WCS and catalog information. Used by *TweakReg*.

Authors
Warren Hack, Mihai Cara

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

class drizzlepac.imgclasses.**Image**(*filename*, *input_catalogs=None*, *exclusions=None*, ***kwargs*)

Bases: object

Primary class to keep track of all WCS and catalog information for a single input image. This class also performs all matching and fitting.

Parameters
filename : str

Filename for image.

input_catalogs : list of str or None

Filename of catalog files for each chip, if specified by user.

kwargs : dict

Parameters necessary for processing derived from input configObj object.

buildDefaultRefWCS()
Generate a default reference WCS for this image.

buildSkyCatalog()
Convert sky catalog for all chips into a single catalog for the entire field-of-view of this image.

clean()
Remove intermediate files created.

close()
Close any open file handles and flush updates to disk

compute_fit_rms()

get_shiftfile_row()
Return the information for a shiftfile for this image to provide compatability with the IRAF-based MultiDrizzle.

get_wcs()
Helper method to return a list of all the input WCS objects associated with this image.

get_xy_catnames()
Return a string with the names of input_xy catalog names

match(*refimage*, *quiet_identity*, ***kwargs*)
Uses xyxymatch to cross-match sources between this catalog and a reference catalog (refCatalog).

openFile(*openDQ=False*)
Open file and set up filehandle for image file

performFit(***kwargs*)
Perform a fit between the matched sources.

Parameters
kwargs : dict

Parameter necessary to perform the fit; namely, *fitgeometry*.

Notes

This task still needs to implement (eventually) interactive iteration of the fit to remove outliers.

sortSkyCatalog()
Sort and clip the source catalog based on the flux range specified by the user. It keeps a copy of the original full list in order to support iteration.

transformToRef(*ref_wcs*, *force=False*)
Transform sky coords from ALL chips into X,Y coords in reference WCS.

updateHeader(*wcsname=None*, *reusename=False*)
Update header of image with shifts computed by *perform_fit()*.

writeHeaderlet(***kwargs*)
Write and/or attach a headerlet based on update to PRIMARY WCS

write_fit_catalog()
Write out the catalog of all sources and resids used in the final fit.

write_outxy(*filename*)
Write out the output(transformed) XY catalog for this image to a file.

write_skycatalog(*filename*)
Write out the all_radec catalog for this image to a file.

class drizzlepac.imgclasses.**RefImage**(*wcs_list*, *catalog*, *xycatalog=None*, *cat_origin=None*, ***kwargs*)
Bases: object

This class provides all the information needed by to define a reference tangent plane and list of source positions on the sky.

Warning: When *wcs_list* is a Python list of *WCS* objects, each element must be an instance of `stwcs.wcsutil.HSTWCS`.

append_not_matched_sources(*image*)

clean()
Remove intermediate files created

clear_dirty_flag()

close()

get_shiftfile_row()
Return the information for a shiftfile for this image to provide compatability with the IRAF-based MultiDrizzle.

set_dirty()

transformToRef()
Transform reference catalog sky positions (self.all_radec) to reference tangent plane (self.wcs) to create output X,Y positions.

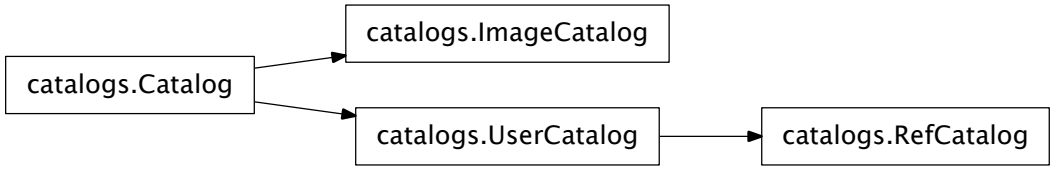
write_skycatalog(*filename*, *show_flux=False*, *show_id=False*)
Write out the all_radec catalog for this image to a file.

13.5 Classes to manage Catalogs and WCS's

This module provides the classes used to generate and manage source catalogs for each input chip. Those positions can be transformed to undistorted sky positions, written out to files, or plotted using various methods defined for these classes.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE



drizzlepac.catalogs.**generateCatalog**(*wcs*, *mode='automatic'*, *catalog=None*, *src_find_filters=None*, ***kwargs*)

Function which determines what type of catalog object needs to be instantiated based on what type of source selection algorithm the user specified.

Parameters

wcs : obj

WCS object generated by STWCS or PyWCS

catalog : str or ndarray

Filename of existing catalog or ndarray of image for generation of source catalog.

kwargs : dict

Parameters needed to interpret source catalog from input catalog with *find-mode* being required.

Returns

catalog : obj

A Catalog-based class instance for keeping track of WCS and associated source catalog

class drizzlepac.catalogs.**ImageCatalog**(*wcs*, *catalog_source*, *src_find_filters=None*, ***kwargs*)

Bases: *drizzlepac.catalogs.Catalog*

Class which generates a source catalog from an image using Python-based, daofind-like algorithms

Required input *kwargs* parameters:

```
computesig, skysigma, threshold, peakmin, peakmax,
hmin, conv_width, [roundlim, sharplim]
```

generateXY(***kwargs*)

Generate source catalog from input image using DAOFIND-style algorithm

class drizzlepac.catalogs.**UserCatalog**(*wcs*, *catalog_source*, ***kwargs*)

Bases: *drizzlepac.catalogs.Catalog*

Class to manage user-supplied catalogs as inputs.

Required input *kwargs* parameters:

```
xyunits, xcol, ycol[, fluxcol, [idcol]]
```

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs : obj

Input WCS object generated using STWCS or HSTWCS

catalog_source : str

Name of the file from which to read the catalog.

kwargs : dict

Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

generateXY(***kwargs*)

Method to interpret input catalog file as columns of positions and fluxes.

plotXYCatalog(***kwargs*)

Plots the source catalog positions using matplotlib's *pyplot.plot()*

Plotting *kwargs* that can also be passed include any keywords understood by matplotlib's *pyplot.plot()* function such as:

```
vmin, vmax, cmap, marker
```

set_colnames()

COLNAMES = ['xcol', 'ycol', 'fluxcol']

IN_UNITS = None

class drizzlepac.catalogs.**RefCatalog**(*wcs*, *catalog_source*, ***kwargs*)

Bases: *drizzlepac.catalogs.UserCatalog*

Class which manages a reference catalog.

Notes

A *reference catalog* is defined as a catalog of undistorted source positions given in RA/Dec which would be used as the master list for subsequent matching and fitting.

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs : obj

Input WCS object generated using STWCS or HSTWCS

catalog_source : str

Name of the file from which to read the catalog.

kwargs : dict

Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

buildXY(*catalogs*)

generateRaDec()

generateXY(***kwargs*)

COLNAMES = ['refxcol', 'refycol', 'rfluxcol']

IN_UNITS = 'degrees'

PAR_PREFIX = 'ref'

class drizzlepac.catalogs.**Catalog**(*wcs*, *catalog_source*, ***kwargs*)
Bases: object

Base class for keeping track of a source catalog for an input WCS

Warning: This class should never be instantiated by itself, as necessary methods are not defined yet.

This class requires the input of a WCS and a source for the catalog, along with any arguments necessary for interpreting the catalog.

Parameters

wcs : obj

Input WCS object generated using STWCS or HSTWCS

catalog_source : str

Name of the file from which to read the catalog.

kwargs : dict

Parameters for interpreting the catalog file or for performing the source extraction from the image. These will be set differently depending on the type of catalog being instantiated.

apply_exclusions(*exclusions*)
Trim sky catalog to remove any sources within regions specified by exclusions file

apply_flux_limits()
Apply any user-specified limits on source selection Limits based on fluxes

buildCatalogs(*exclusions=None*, ***kwargs*)
Primary interface to build catalogs based on user inputs.

generateRaDec()
Convert XY positions into sky coordinates using STWCS methods

generateXY(***kwargs*)
Method to generate source catalog in XY positions Implemented by each subclass

plotXYCatalog(***kwargs*)
Method which displays the original image and overlays the positions of the detected sources from this image's catalog.

Plotting *kwargs* that can be provided are:

vmin, vmax, cmap, marker

Default colormap is *summer*.

set_colnames()
Method to define how to interpret a catalog file Only needed when provided a source catalog as input

writeXYCatalog(*filename*)
Write out the X,Y catalog to a file

PAR_PREFIX = ' '

13.6 Functions to Manage WCS Table Extension

These functions provide the basic support for initializing, creating and updating the WCS table extension which serves as the archive of updates made to the WCS information in the image headers.

stwcs.wcsutil.wcscorr.**archive_wcs_file**(*image*, *wcs_id=None*)
Update WCSCORR table with rows for each SCI extension to record the newly updated WCS keyword values.

stwcs.wcsutil.wcscorr.**create_wcscorr**(*descrip=False*, *numrows=1*, *padding=0*)
Return the basic definitions for a WCSCORR table. The dtype definitions for the string columns are set to the maximum allowed so that all new elements will have the same max size which will be automatically truncated to this limit upon updating (if needed).

The table is initialized with rows corresponding to the OPUS solution for all the 'SCI' extensions.

stwcs.wcsutil.wcscorr.**delete_wcscorr_row**(*wcstab*, *selections=None*, *rows=None*)
Sets all values in a specified row or set of rows to default values

This function will essentially erase the specified row from the table without actually removing the row from the table. This avoids the problems with trying to resize the number of rows in the table while preserving the ability to update the table with new rows again without resizing the table.

Parameters

wcstab: object

PyFITS binTable object for WCSCORR table

selections: dict

Dictionary of wcscorr column names and values to be used to select the row or set of rows to erase

rows: int, list

If specified, will specify what rows from the table to erase regardless of the value of 'selections'

stwcs.wcsutil.wcscorr.**find_wcscorr_row**(*wcstab*, *selections*)

Return an array of indices from the table (NOT HDU) 'wcstab' that matches the selections specified by the user.

The row selection criteria must be specified as a dictionary with column name as key and value(s) representing the valid desired row values. For example, {'wcs_id':'OPUS','extver':2}.

stwcs.wcsutil.wcscorr.**init_wcscorr**(*input*, *force=False*)

This function will initialize the WCSCORR table if it is not already present, and look for WCS keywords with a prefix of 'O' as the original OPUS generated WCS as the initial row for the table or use the current WCS keywords as initial row if no 'O' prefix keywords are found.

This function will NOT overwrite any rows already present.

This function works on all SCI extensions at one time.

stwcs.wcsutil.wcscorr.**restore_file_from_wcscorr**(*image*, *id='OPUS'*, *wcskey=''*)

Copies the values of the WCS from the WCSCORR based on ID specified by user. The default will be to restore the original OPUS-derived values to the Primary WCS. If wcskey is specified, the WCS with that key will be updated instead.

stwcs.wcsutil.wcscorr.**update_wcscorr**(*dest*, *source=None*, *extname='SCI'*, *wcs_id=None*, *active=True*)

Update WCSCORR table with a new row or rows for this extension header. It copies the current set of WCS keywords as a new row of the table based on keyed WCSs as per Paper I Multiple WCS standard).

Parameters

dest : HDUList

The HDU list whose WCSCORR table should be appended to (the WCSCORR HDU must already exist)

source : HDUList, optional

The HDU list containing the extension from which to extract the WCS keywords to add to the WCSCORR table. If None, the dest is also used as the source.

extname : str, optional

The extension name from which to take new WCS keywords. If there are multiple extensions with that name, rows are added for each extension version.

wcs_id : str, optional

The name of the WCS to add, as in the WCSNAMEa keyword. If unspecified, all the WCSs in the specified extensions are added.

active: bool, optional

When True, indicates that the update should reflect an update of the active WCS information, not just appending the WCS to the file as a headerlet

stwcs.wcsutil.wcscorr.**update_wcscorr_column**(*wcstab*, *column*, *values*, *selections=None*, *rows=None*)

Update the values in 'column' with 'values' for selected rows

Parameters

wcstab: object

PyFITS binTable object for WCSCORR table

column: string

Name of table column with values that need to be updated

values: string, int, or list

Value or set of values to copy into the selected rows for the column

selections: dict

Dictionary of wcscorr column names and values to be used to select the row or set of rows to erase

rows: int, list

If specified, will specify what rows from the table to erase regardless of the value of 'selections'

13.7 Functions to Manage Legacy OPUS WCS Keywords in the WCS Table

The previously released versions of `makewcs` provided with *MultiDrizzle archives* the original OPUS generated WCS keywords using header keywords which have a prefix of "O", such as "OCRPIX1". In order to avoid overwriting or ignoring these original values, these functions can be used to convert the prefixed OPUS WCS keywords into WCS table entries compatible with the new code.

Strictly to provide complete support for these OPUS keywords, the code will also create, if the user desires, prefix "O" WCS keywords from the alternate WCS FITS conventions OPUS keywords. This would allow images processed using the new code only can then be used with older versions of *MultiDrizzle*, if the user needs such compatibility.

`stwcs.wcsutil.convertwcs.`**`archive_prefix_OPUS_WCS`**(*fobj*, *extname='SCI'*)

Identifies WCS keywords which were generated by OPUS and archived using a prefix of 'O' for all 'SCI' extensions in the file

Parameters

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

Filename or fits object of a file

`stwcs.wcsutil.convertwcs.`**`create_prefix_OPUS_WCS`**(*fobj*, *extname='SCI'*)

Creates alternate WCS with a prefix of 'O' for OPUS generated WCS values to work with old MultiDrizzle.

Parameters

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

Filename or fits object of a file

Raises

IOError:

if input FITS object was not opened in 'update' mode

13.8 TWEAKUTILS: Utility Functions for Tweakreg

The functions in this module support the various aspects of *tweakreg*, including finding the objects in the images and plotting the residuals.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

`drizzlepac.tweakutils.`**`build_pos_grid`**(*start*, *end*, *nstep*, *mesh=False*)

Return a grid of positions starting at X,Y given by 'start', and ending at X,Y given by 'end'. The grid will be completely filled in X and Y by every 'step' interval.

`drizzlepac.tweakutils.`**`build_xy_zeropoint`**(*imgxy*, *refxy*, *searchrad=3.0*, *histplot=False*, *figure_id=1*, *plotname=None*, *interactive=True*)

Create a matrix which contains the delta between each XY position and each UV position.

`drizzlepac.tweakutils.`**`createWcsHDU`**(*wcs*)

Generate a WCS header object that can be used to populate a reference WCS HDU.

For most applications, stwcs.wcsutil.HSTWCS.wcs2header() will work just as well.

drizzlepac.tweakutils.**find_xy_peak**(*img*, *center=None*, *sigma=3.0*)
Find the center of the peak of offsets

drizzlepac.tweakutils.**gauss**(*x*, *sigma*)
Compute 1-D value of gaussian at position x relative to center.

drizzlepac.tweakutils.**gauss_array**(*nx*, *ny=None*, *fwhm=1.0*, *sigma_x=None*, *sigma_y=None*, *zero_norm=False*)
Computes the 2D Gaussian with size nx*ny.

Parameters
nx : int

ny : int [Default: None]

Size of output array for the generated Gaussian. If ny == None, output will be an array nx X nx pixels.

fwhm : float [Default: 1.0]

Full-width, half-maximum of the Gaussian to be generated

sigma_x : float [Default: None]

sigma_y : float [Default: None]

Sigma_x and sigma_y are the stddev of the Gaussian functions.

zero_norm : bool [Default: False]

The kernel will be normalized to a sum of 1 when True.

Returns
gauss_arr : array

A numpy array with the generated gaussian function

drizzlepac.tweakutils.**isfloat**(*value*)
Return True if all characters are part of a floating point value

drizzlepac.tweakutils.**make_vector_plot**(*coordfile, columns=[1, 2, 3, 4], data=None, figure_id=None, title=None, axes=None, every=1, labelsize=8, ylimit=None, limit=None, xlower=None, ylower=None, output=None, headl=4, headw=3, xsh=0.0, ysh=0.0, fit=None, scale=1.0, vector=True, textscale=5, append=False, linfit=False, rms=True, plotname=None*)
Convert a XYXYMATCH file into a vector plot or set of residuals plots.

This function provides a single interface for generating either a vector plot of residuals or a set of 4 plots showing residuals. The data being plotted can also be adjusted for a linear fit on-the-fly.

Parameters
coordfile : string

Name of file with matched sets of coordinates. This input file can be a file compatible for use with IRAF's geomap.

columns : list [Default: [0,1,2,3]]

Column numbers for the X,Y positions from each image

data : list of arrays

If specified, this can be used to input matched data directly

title : string

Title to be used for the generated plot

axes : list

List of X and Y min/max values to customize the plot axes

every : int [Default: 1]

Slice value for the data to be plotted

limit : float

Radial offset limit for selecting which sources are included in the plot

labelsize : int [Default: 8] or str

Font size to use for tick labels, either in font points or as a string understood by tick_params().

ylimit : float

Limit to use for Y range of plots.

xlower : float

ylower : float

Limit in X and/or Y offset for selecting which sources are included in the plot

output : string

Filename of output file for generated plot

headl : int [Default: 4]

Length of arrow head to be used in vector plot

headw : int [Default: 3]

Width of arrow head to be used in vector plot

xsh : float

ysh : float

Shift in X and Y from linear fit to be applied to source positions from the first image

scale : float

Scale from linear fit to be applied to source positions from the first image

fit : array

Array of linear coefficients for rotation (and scale?) in X and Y from a linear fit to be applied to source positions from the first image

vector : bool [Default: True]

Specifies whether or not to generate a vector plot. If False, task will generate a set of 4 residuals plots instead

textscale : int [Default: 5]

Scale factor for text used for labelling the generated plot

append : bool [Default: False]

If True, will overplot new plot on any pre-existing plot

linfit : bool [Default: False]

If True, a linear fit to the residuals will be generated and added to the generated residuals plots

rms : bool [Default: True]

Specifies whether or not to report the RMS of the residuals as a label on the generated plot(s).

plotname : str [Default: None]

Write out plot to a file with this name if specified.

drizzlepac.tweakutils.**ndfind_old**(*array, hmin, fwhm, sharplim=[0.2, 1.0], roundlim=[-1, 1], minpix=5, datamax=None*)

Source finding algorithm based on NDIMAGE routines

This function provides a simple replacement for the DAOFIND task.

Parameters

array : arr

Input image as numpy array

hmin : float

Limit for source detection in pixel values

fwhm : float

Full-width half-maximum of the PSF in the image

minpix : int

Minimum number of pixels for any valid source

sharplim : tuple

[Not used at this time]

roundlim : tuple

[Not used at this time]

datamax : float

Maximum good pixel value found in any detected source

Returns

x : arr

Array of detected source X positions (in array coordinates, 0-based)

y : arr

Array of detected source Y positions (in array coordinates, 0-based)

flux : arr

Array of detected source fluxes in pixel values

id : arr

Array of detected source ID numbers

drizzlepac.tweakutils.**parse_atfile_cat**(*input*)

Return the list of catalog filenames specified as part of the input @-file

drizzlepac.tweakutils.**parse_colname**(*colname*)

Common function to interpret input column names provided by the user.

This function translates column specification provided by the user into a column number.

Parameters

colname :

Column name or names to be interpreted

Returns

cols : list

The return value will be a list of strings.

Notes

This function will understand the following inputs:

```
'1,2,3' or   'c1,c2,c3' or ['c1','c2','c3']
'1-3'   or   'c1-c3'
'1:3'   or   'c1:c3'
'1 2 3' or   'c1 c2 c3'
'1'     or   'c1'
1
```

drizzlepac.tweakutils.**parse_exclusions**(*exclusions*)

Read in exclusion definitions from file named by 'exclusions' and return a list of positions and distances

drizzlepac.tweakutils.**parse_skypos**(*ra*, *dec*)

Function to parse RA and Dec input values and turn them into decimal degrees

Input formats could be:

["nn","nn","nn.nn"] "nn nn nn.nnn" "nn:nn:nn.nn" "nnH nnM nn.nnS" or "nnD nnM nn.nnS" nn.nnnnnnnn "nn.nnnnnnn"

drizzlepac.tweakutils.**plot_zeropoint**(*pars*)

Plot 2d histogram.

Pars will be a dictionary containing:

data, figure_id, vmax, title_str, xp,yp, searchrad

drizzlepac.tweakutils.**radec_hmstodd**(*ra*, *dec*)

Function to convert HMS values into decimal degrees.

This function relies on the astropy.coordinates package to perform the conversion to decimal degrees.

Parameters

ra : list or array

List or array of input RA positions

dec : list or array

List or array of input Dec positions

Returns

pos : arr

Array of RA,Dec positions in decimal degrees

See also:

`astropy.coordinates`

Notes

This function supports any specification of RA and Dec as HMS or DMS; specifically, the formats:

```
["nn","nn","nn.nn"]
"nn nn nn.nnn"
"nn:nn:nn.nn"
"nnH nnM nn.nnS" or "nnD nnM nn.nnS"
```

drizzlepac.tweakutils.**read_ASCII_cols**(*infile, cols=[1, 2, 3]*)

Interpret input ASCII file to return arrays for specified columns.

Returns

outarr : list of arrays

The return value will be a list of numpy arrays, one for each 'column'.

Notes

The specification of the columns should be expected to have lists for each 'column', with all columns in each list combined into a single entry. For example:

```
    cols = ['1,2,3','4,5,6',7]
```

where '1,2,3' represent the X/RA values, '4,5,6' represent the Y/Dec values and 7 represents the flux value for a total of 3 requested columns of data to be returned.

drizzlepac.tweakutils.**read_FITS_cols**(*infile*, *cols=None*)
Read columns from FITS table

drizzlepac.tweakutils.**readcols**(*infile*, *cols=None*)
Function which reads specified columns from either FITS tables or ASCII files

This function reads in the columns specified by the user into numpy arrays regardless of the format of the input table (ASCII or FITS table).

Parameters
infile : string

Filename of the input file

cols : string or list of strings

Columns to be read into arrays

Returns
outarr : array

Numpy array or arrays of columns from the table

drizzlepac.tweakutils.**write_shiftfile**(*image_list*, *filename*, *outwcs='tweak_wcs.fits'*)
Write out a shiftfile for a given list of input Image class objects

13.9 UPDATEHDR: Functions for Updating WCS with New Solutions

The functions in this module support updating the WCS information in distorted images with the alignment solution determined by *tweakreg* or saved in a shiftfile.

Authors
Warren Hack, Mihai Cara

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.updatehdr.**create_unique_wcsname**(*fimg*, *extnum*, *wcsname*)
This function evaluates whether the specified wcsname value has already been used in this image. If so, it automatically modifies the name with a simple version ID using wcsname_NNN format.

Parameters
fimg : obj

PyFITS object of image with WCS information to be updated

extnum : int

Index of extension with WCS information to be updated

wcsname : str

Value of WCSNAME specified by user for labelling the new WCS

Returns

uniqname : str

Unique WCSNAME value

drizzlepac.updatehdr.**update_from_shiftfile**(*shiftfile*, *wcsname=None*, *force=False*)

Update headers of all images specified in shiftfile with shifts from shiftfile.

Parameters

shiftfile : str

Filename of shiftfile.

wcsname : str

Label to give to new WCS solution being created by this fit. If a value of None is given, it will automatically use 'TWEAK' as the label. [Default =None]

force : bool

Update header even though WCS already exists with this solution or wcsname? [Default=False]

drizzlepac.updatehdr.**update_wcs**(*image*, *extnum*, *new_wcs*, *wcsname=''*, *reusename=False*, *verbose=False*)

Updates the WCS of the specified extension number with the new WCS after archiving the original WCS.

The value of 'new_wcs' needs to be the full HSTWCS object.

Parameters

image : str

Filename of image with WCS that needs to be updated

extnum : int

Extension number for extension with WCS to be updated/replaced

new_wcs : object

Full HSTWCS object which will replace/update the existing WCS

wcsname : str

Label to give newly updated WCS

reusename : bool

User can choose whether to over-write WCS with same name or not. [Default: False]

verbose : bool, int

Print extra messages during processing? [Default: False]

drizzlepac.updatehdr.**updatewcs_with_shift**(*image*, *reference*, *wcsname=None*, *reusename=False*, *fitgeom='rscale'*, *rot=0.0*, *scale=1.0*, *xsh=0.0*, *ysh=0.0*, *fit=None*, *xrms=None*, *yrms=None*, *verbose=False*, *force=False*, *sciext='SCI'*)

Update the SCI headers in 'image' based on the fit provided as determined in the WCS specified by 'reference'. The fit should be a 2-D matrix as generated for use with 'make_vector_plot()'.

Parameters

image : str or PyFITS.HDUList object

Filename, or PyFITS object, of image with WCS to be updated. All extensions with EXTNAME matches the value of the 'sciext' parameter value (by default, all 'SCI' extensions) will be updated.

reference : str

Filename of image/headerlet (FITS file) which contains the WCS used to define the tangent plane in which all the fit parameters (shift, rot, scale) were measured.

wcsname : str

Label to give to new WCS solution being created by this fit. If a value of None is given, it will automatically use 'TWEAK' as the label. If a WCS has a name with this specific value, the code will automatically append a version ID using the format '_n', such as 'TWEAK_1', 'TWEAK_2',or 'TWEAK_update_1'. [Default =None]

reusename : bool

User can specify whether or not to over-write WCS with same name. [Default: False]

rot : float

Amount of rotation measured in fit to be applied. [Default=0.0]

scale : float

Amount of scale change measured in fit to be applied. [Default=1.0]

xsh : float

Offset in X pixels from defined tangent plane to be applied to image. [Default=0.0]

ysh : float

Offset in Y pixels from defined tangent plane to be applied to image. [Default=0.0]

fit : arr

Linear coefficients for fit [Default = None]

xrms : float

RMS of fit in RA (in decimal degrees) that will be recorded as CRDER1 in WCS and header [Default = None]

yrms : float

RMS of fit in Dec (in decimal degrees) that will be recorded as CRDER2 in WCS and header [Default = None]

verbose : bool

Print extra messages during processing? [Default=False]

force : bool

Update header even though WCS already exists with this solution or wcsname? [Default=False]

sciext : string

Value of FITS EXTNAME keyword for extensions with WCS headers to be updated with the fit values. [Default='SCI']

Notes

The algorithm used to apply the provided fit solution to the image involves applying the following steps to the WCS of each of the input image's chips:

1. **compute RA/Dec with full distortion correction for** reference point as (Rc_i,Dc_i)
2. **find the Xc,Yc for each Rc_i,Dc_i and get the difference from the** CRPIX position for the reference WCS as (dXc_i,dYc_i)
3. **apply fit (rot&scale) to (dXc_i,dYc_i) then apply shift, then add** CRPIX back to get new (Xcs_i,Ycs_i) position
4. compute (Rcs_i,Dcs_i) as the sky coordinates for (Xcs_i,Ycs_i)
5. compute delta of (Rcs_i-Rc_i, Dcs_i-Dcs_i) as (dRcs_i,dDcs_i)
6. **apply the fit to the chip's undistorted CD matrix, the apply linear** distortion terms back in to create a new CD matrix
7. add (dRcs_i,dDcs_i) to CRVAL of the reference chip's WCS
8. update header with new WCS values

13.10 Region mapping for TweakReg

This module provides functions for mapping DS9 region files given in sky coordinates to DS9 region files specified in image coordinates of multiple images using the WCS information from the images.

Authors

Mihai Cara

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.mapreg.**MapReg**(*input_reg*, *images*, *img_wcs_ext='sci'*, *refimg=''*, *ref_wcs_ext='sci'*, *chip_reg=''*, *outpath='./regions'*, *filter=''*, *catfname=''*, *iteractive=False*, *append=False*, *verbose=True*)

`MapReg` provides an automated interface for converting a region file to the image coordinate system (CS) of multiple images (and their extensions) using WCS information from the image(s) header(s). This conversion does not take into account pointing errors and, therefore, an examination and adjustment (if required) of output region files is highly recommended. This task is designed to simplify the creation of the exclusions and/or inclusions region files used with `TweakReg` task for sources finding.

Parameters

input_reg : string or list of strings (Default = '')

Input region files that need to be mapped to image CS using WCS information from *images* (see below). Only region files saved in sky CS are allowed in this release. Regions specified in image-like coordinates (e.g., image, physical) will be ignored.

This paramater can be provided in any of several forms:

- filename of a single image
- comma-separated list of filenames
- `@-file` filelist containing list of desired input region filenames

The `@-file` filelist needs to be provided as an ASCII text file containing a list of filenames for all input region files with one filename on each line of the file.

images : string or list of strings (Default = `*.fits`)

FITS images onto which the region files *input_reg* will be mapped. These image files must contain WCS information in their headers in order to convert *input_reg* from sky coordinates to correct image coordinates. This paramater can be provided in any of several forms:

- filename of a single image
- filename of an association (ASN)table
- wild-card specification for files in directory (using `*`, `?` etc.)
- comma-separated list of filenames
- `@-file` filelist containing list of desired input filenames (and optional inverse variance map filenames)

The `@-file` filelist needs to be provided as an ASCII text file containing a list of filenames for all input images (to which *input_reg* regions should be mapped) with one filename on each line of the file.

img_wcs_ext : string or list of strings (Default = `SCI`)

Extension name, extension name and version, or extension number of FITS extensions in the *images* to which the input regions *input_reg* should be mapped. The header of each extension must contain WCS information that will be used to convert *input_reg* from sky CS to image-like CS. Multiple extensions must be separated by semicolon while extension name and version (if present) must be separated by comma, e.g., `'SCI;DQ,1;0'`. When specifying the extension name only, internally it will be expanded into a list of extension names and versions for each version of that extension name present in the input `images`. For example, if a FITS file has four SCI and four DQ extensions, then `'SCI;DQ,1;0'` will be expanded into `'SCI,1;SCI,2;SCI,3;SCI,4;DQ,1;0'`.

refimg : string (Default = '')

Reserved for future use. Filename of the reference image. May contain extension specifier: [extname,extver], [extname], or [extnumber].

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

ref_wcs_ext : string (Default = `SCI`)

Reserved for future use. Extension name and/or version of FITS extensions in the `refimg` that contain WCS information that will be used to convert *input_reg* from image-like CS to sky CS. NOTE: Only extension name is allowed when *input_reg* is a list of region files that contain regions in image-like CS. In this case, the number of regions in *input_reg* must agree with the number of extensions with name specified by `ref_wcs_ext` present in the `refimg` FITS image.

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

chip_reg : string or list of strings (Default = '')

Input region files in image CS associated with each extension specified by the `img_wcs_ext` parameter above. These regions will be added directly (without any transformation) to the *input_reg* regions mapped to each extension of the input `images`. These regions must be specified in image-like coordinates. Typically, these regions should contain "exclude" regions to exclude parts of the image specific to the detector **chip** (e.g., vignetted regions due to used filters, or occulting finger in ACS/HRC images) from being used for source finding. This paramater can be provided in one of the following forms:

- filename of a single image (if `img_wcs_ext` specifies a single FITS extension);

- comma-separated list of filenames (if `img_wcs_ext` specifies more than one extension) or *None* for extensions that do not need any chip-specific regions to be excluded/included;
- '' (empty string) or None if no chip-specific region files are provided.

The number of regions ideally must be equal to the number of extensions specified by the `img_wcs_ext` parameter. If the number of chip-specific regions is less than the number of `img_wcs_ext` extensions then 'chip_reg' regions will be assigned to the first extensions from `img_wcs_ext` (after internal expansion described in help for the `img_wcs_ext` parameter above). If the number of 'chip_reg' is larger than the number of `img_wcs_ext` extensions then extra regions will be ignored.

outpath : string (Default = `./regions`)

The directory to which the transformed regions should be saved.

filter : string {'None', 'fast', or 'precise' } (Default = 'None')

Specify whether or not to remove the regions in the transformed region files that are outside the image array. With the 'fast' mehod only intersection of the bounding boxes is being checked.

Note: The `'precise'` method is not implemented in this release and, if specified, defaults to 'fast'. The `'precise'` option is not available through the `TEAL` interface.

---

catfname : string (Default = `exclusions_cat.txt`)

The file name of the output exclusions catalog file to be created from the supplied image and region file names. This file can be passed as an input to TweakReg task. Verify that the created file is correct!

append : bool (Default = False)

Specify whether or not to append the transformed regions to the existing region files with the same name.

interactive : bool (Default = False)

Reserved for future use. (This switch controls whether the program stops and waits for the user to examine any generated region files before continuing on to the next image.)

Note: This parameter is reserved for future use and it is not available through `TEAL` interface.

---

verbose : bool (Default = False)

Specify whether or not to print extra messages during processing.

Notes

NOTE 1: This task takes a region file (or multiple files) that describe(s) what regions of sky should be used for source finding (*include* regions) and what regions should be avoided (*exclude* regions) and transforms/maps this region file onto a number of image files that need to be alighned.

The idea behind this task is automate the creation of region files that then can be passed to *exclusions* parameter of the *TweakReg* task.

The same thing can be achieved manually using, for example, external FITS viewers, e.g., SAO DS9. For example, based on some image `refimg.fits` we can select a few small regions of sky that contain several good (bright, not saturated) point-like sources that could be used for image alignment of other images (say `img1.fits`, `img2.fits`, etc.). We can save this region file in sky coordinates (e.g., `fk5`), e.g., under the name `input_reg.reg`. We can then load a specific extension of each of the images `img1.fits`, `img2.fits`, etc. one by one into DS9 and then load onto those images the previously created include/exclude region file `input_reg.reg`. Now we can save the regions using *image* coordinates. To do conversion from the sky coordinates to image coordinates, DS9 will use the WCS info from the image onto which the region file was loaded. The `MapReg` task tries to automate this process.

NOTE 2: `MapReg` relies on the `pyregion` package for region file parsing and coordinate transformation. Unfortunately, as of writing, **pyregion** does not consider distortion corrections when performing coordinate transformations. Therefore, there might be a slight discrepancy between the regions produced by `MapReg` and the DS9 regions obtained as described in the NOTE 1 above.

NOTE 3: `MapReg` does not take into account pointing errors and thus the produced region files can be somewhat misaligned compared to their intended position around the sources identified in the "reference" image. Threfore, it is highly recommended that the produced region files be loaded into DS9 and their position be adjusted manually to include the sources of interest (or to avoid the regions that need to be avoided). If possible, the *include* or *exclude* regions should be large enough as to allow for most pointing errors.

Examples

Let's say that based on some image `refimg.fits` we have produced a "master" reference image (`master.reg`) that includes regions around sources that we want to use for image alignment in task `TweakReg` and excludes regions that we want to avoid being used for image alignment (e.g, diffraction spikes, saturated quasars, stars, etc.). We save the file `master.reg` in sky CS (e.g., `fk5`).

Also, let's assume that we have a set of images `img1.fits`, `img2.fits`, etc. with four FITS extensions named 'SCI' and 'DQ'. For some of the extensions, after analizing the `img*.fits` images we have identified parts of the chips that cannot be used for image alighnment. We create region files for those extensions and save the files in image CS as, e.g., `img1_chip_sci2.reg` (in our example this will be the only chip that needs "special" treatment).

Finally, let's say we want to "replicate" the "master" region file to all SCI exctensions of the `img*.fits` images as well as to the 2nd DQ extension and to the 8th extension of the `img*.fits` images.

To do this we run:

```
>>> mapreg(input_reg = 'master.reg', images='img*.fits',
...         img_wcs_ext='sci;dq,2;8', chip_reg='None,
...         img1_chip_sci2.reg,None,None,None,None')
```

This will produce six region files in the ./regions subdirectory for *each* input image:

```
``img1_sci1_twreg.reg``,    ``img1_sci2_twreg.reg``,    ``img1_sci3_twreg.reg``,
``img1_sci4_twreg.reg``,    ``img1_dq2_twreg.reg``,     ``img1_extn8_twreg.reg``
...
```

```
``img2_sci1_twreg.reg``,    ``img2_sci2_twreg.reg``,    ``img2_sci3_twreg.reg``,
``img2_sci4_twreg.reg``,    ``img2_dq2_twreg.reg``,     ``img2_extn8_twreg.reg``
...
```

drizzlepac.mapreg.**map_region_files**(*input_reg*, *images*, *img_wcs_ext='sci'*, *refimg=None*, *ref_wcs_ext='sci'*, *chip_reg=None*, *outpath='./regions'*, *filter=None*, *catfname=None*, *iteractive=False*, *append=False*, *verbose=True*)

drizzlepac.mapreg.**help**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

13.11 Photometric equalization for AstroDrizzle

A tool to adjust data values of images by equalizing each chip's PHOTFLAM value to a single common value so that all chips can be treated equally by *AstroDrizzle*.

Authors

Mihai Cara

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

drizzlepac.photeq.**photeq**(*files='*_flt.fits'*, *sciext='SCI'*, *errext='ERR'*, *ref_phot=None*, *ref_phot_ext=None*, *phot_kwd='PHOTFLAM'*, *aux_phot_kwd='PHOTFNU'*, *search_primary=True*, *readonly=True*, *clobber=False*, *logfile='photeq.log'*)

Adjust data values of images by equalizing each chip's PHOTFLAM value to a single common value so that all chips can be treated equally by *AstroDrizzle*.

Parameters

files : str (Default = `'*_flt.fits'`)

A string containing one of the following:

- a comma-separated list of valid science image file names, e.g.: `'j1234567q_flt.fits, j1234568q_flt.fits'`;
- an @-file name, e.g., `'@files_to_match.txt'`. See notes section for details on the format of the @-files.

Note: Valid science image file names are:

- file names of existing FITS, GEIS, or WAIVER FITS files;
- partial file names containing wildcard characters, e.g., `'*_flt.fits'`;
- Association (ASN) tables (must have `_asn`, or `_asc` suffix), e.g., `'j12345670_asn.fits'`.

---

sciext : str (Default = 'SCI')

Extension *name* of extensions whose data and/or headers should be corrected.

errext : str (Default = 'ERR')

Extension *name* of the extensions containing corresponding error arrays. Error arrays are corrected in the same way as science data.

ref_phot : float, None (Default = None)

A number indicating the new value of PHOTFLAM or PHOTFNU (set by 'phot_kwd') to which the data should be adjusted.

ref_phot_ext : int, str, tuple, None (Default = None)

Extension from which the *photeq* should get the reference photometric value specified by the *phot_kwd* parameter. This parameter is ignored if *ref_phot* **is not** *None*. When *ref_phot_ext* is *None*, then the reference inverse sensitivity value will be picked from the first *sciext* of the first input image containing *phot_kwd*.

phot_kwd : str (Default = 'PHOTFLAM')

Specifies the primary keyword which contains inverse sensitivity (e.g., PHOTFLAM). It is used to compute conversion factors by which data should be rescaled.

aux_phot_kwd : str, None, list of str (Default = 'PHOTFNU')

Same as *phot_kwd* but describes *other* photometric keyword(s) that should be corrected by inverse of the scale factor used to correct data. These keywords are *not* used to compute conversion factors. Multiple keywords can be specified as a Python list of strings: `['PHOTFNU', 'PHOTOHMY']`.

Note: If specifying multiple secondary photometric keywords in the TEAL interface, use a comma-separated list of keywords.

---

search_primary : bool (Default = True)

Specifies whether to first search the primary header for the presence of *phot_kwd* keyword and compute conversion factor based on that value. This is (partially) ignored when *ref_phot* is not *None* in the sense that the value specified by *ref_phot* will be used as the reference *but* in all images primary will be searched for *phot_kwd* and *aux_phot_kwd* and those values will be corrected (if `search_primary=True`).

readonly : bool (Default = True)

If *True*, *photeq* will not modify input files (nevertheless, it will convert input GEIS or WAVERED FITS files to MEF and could overwrite existing MEF files if *clobber* is set to *True*). The (console or log file) output however will be identical to the case when `readonly=False` and it can be examined before applying these changes to input files.

clobber : bool (Default = False)

Overwrite existing MEF files when converting input WAVERED FITS or GEIS to MEF.

logfile : str, None (Default = 'photeq.log')

File name of the log file.

Notes

By default, *photeq* will search for the first inverse sensitivity value (given by the header keyword specified by the *phot_kwd* parameter, e.g., PHOTFLAM or PHOTFNU) found in the input images and it will equalize all other images to this reference value.

It is possible to tell *photeq* to look for the reference inverse sensitivity value only in a specific extension of input images, e.g.: 3, ('sci',3), etc. This can be done by setting *ref_phot_ext* to a specific extension. This may be useful, for example, for WFPC2 images: WF3 chip was one of the better calibrated chips, and so, if one prefers to have inverse sensitivities equalized to the inverse sensitivity of the WF3 chip, one can set `ref_phot_ext=3`.

Alternatively, one can provide their own reference inverse sensitivity value to which all other images should be "equalized" through the parameter *ref_phot*.

Note: Default parameter values (except for *files*, *readonly*, and *clobber*) should be acceptable for most HST images.

Warning: If images are intended to be used with *AstroDrizzle*, it is recommended that sky background measurement be performed on "equalized" images as the *photeq* is not aware of sky user keyword in the image headers and thus it cannot correct sky values already recorded in the headers.

Examples

1.In most cases the default parameters should suffice:

```
>>> from drizzlepac import photeq
>>> photeq.photeq(files='*_flt.fits', readonly=False)
```

2.If the re-calibration needs to be done on PHOTFNU rather than PHOTFLAM, then:

```
>>> photeq.photeq(files='*_flt.fits', ref_phot='PHOTFNU',
... aux_phot_kwd='PHOTFLAM')
```

3.If for WFPC2 data one desires that PHOTFLAM from WF3 be used as the reference in WFPC2 images, then:

```
>>> photeq.photeq(files='*_flt.fits', ref_phot_ext=3) # or ('sci',3)
```

drizzlepac.photeq.**help**(*file=None*)

Print out syntax help for running skymatch

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

CHAPTER

FOURTEEN

COORDINATE TRANSFORMATION TASKS

These tasks support transformations of source positions to and from distorted and drizzled images.

14.1 pixtopix: Coordinate transformation to/from drizzled images

This task allows a user to perform coordinate transformations with the full WCS and distortion model to and from drizzled image positions. This task serves as a replacement for the STSDAS.dither task 'tran'. *pixtopix* transforms pixel positions given as X,Y values into positions in another WCS frame based on the WCS information and any recognized distortion keywords from the input image header.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

14.1.1 Parameters

inimage
[str] full filename with path of input image, an extension name `['sci',1]` should be provided if input is a multi-extension `FITS` file

outimage
[str, optional] full filename with path of output image, an extension name `['sci',1]` should be provided if output is a multi-extension `FITS` file. If no image gets specified, the input image will be used to generate a default output WCS using `stwcs.distortion.util.output_wcs`.

direction
[str] Direction of transform (forward or backward). The 'forward' transform takes the pixel positions (assumed to be from the 'input' image) and determines their position in the 'output' image. The 'backward' transform converts the pixel positions (assumed to be from the 'output' image) into pixel positions in the 'input' image.

14.1.2 Optional Parameters

x
[float, optional] X position from image

y
[float, optional] Y position from image

coords
[str, deprecated] [DEPRECATED] full filename with path of file with x,y coordinates Filename given here will be *ignored* if a file has been specified in *coordfile* parameter.

coordfile
[str, optional] full filename with path of file with starting x,y coordinates

colnames
[str, optional] comma separated list of column names from 'coords' files containing x,y coordinates, respectively. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',... convention.

separator
[str, optional] non-blank separator used as the column delimiter in the coords file

precision
[int, optional] Number of floating-point digits in output values

output
[str, optional] Name of output file with results, if desired

verbose
[bool] Print out full list of transformation results (default: False)

14.1.3 RETURNS

outx
[float] X position of transformed pixel. If more than 1 input value, then it will be a numpy array.

outy
[float] Y position of transformed pixel. If more than 1 input value, then it will be a numpy array.

14.1.4 Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with AstroDrizzle. Input images can be updated to use these conventions through the use of the 'updatewcs' module the *STWCS* package.

14.1.5 See Also

`stwcs`

14.1.6 Examples

This task can be run from either the TEAL GUI or from the Python command-line. These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. Run the task using the TEAL GUI under PyRAF:

```
>>> import drizzlepac
>>> epar pixtopix
```

2. Convert the position 256,256 from 'input_flt.fits[sci,1]' into a position on the output image 'output_drz.fits[sci,1]' using:

```
>>> from drizzlepac import pixtopix
>>> outx,outy = pixtopix.tran("input_file_flt.fits[sci,1]",
...                           "output_drz.fits[sci,1],"forward", 256,256)
```

   or if you the default direction of 'forward' is already appropriate and you don't want to explicitly set that value:

```
>>> outx,outy = pixtopix.tran("input_file_flt.fits[sci,1]",
...                           "output_drz.fits[sci,1],x=256,y=256)
```

3. The set of X,Y positions from 'output_drz.fits[sci,1]' stored as the 3rd and 4th columns from the ASCII file `xy_sci1.dat` will be transformed into pixel positions from 'input_flt.fits[sci,1]' and written out to `xy_flt1.dat` using:

```
>>> from drizzlepac import pixtopix
>>> x,y = pixtopix.tran("input_flt.fits[sci,1]",
...          "output_drz.fits[sci,1]", "backward",
...          coordfile='xy_sci1.dat', colnames=['c3','c4'],
...          output="xy_flt1.dat")
```

`drizzlepac.pixtopix.`**`tran`**(*inimage*, *outimage*, *direction='forward'*, *x=None*, *y=None*, *coords=None*, *coordfile=None*, *colnames=None*, *separator=None*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations in pixel coordinates between 2 images using STWCS and full distortion models read from each image's header.

14.2 pixtosky: Coordinate transformation to sky coordinates

This task allows a user to perform coordinate transformations with the full WCS and distortion model on source positions from an input image to sky coordinates. This task serves as a replacement for the IRAF.STSDAS task *xy2rd*, albeit with the added capability of understanding the full distortion model provided in the headers of images updated for use with astrodrizzle and tweakreg. *pixtosky* transforms pixel

positions given as X,Y values into positions on the sky based on the WCS information and any recognized distortion keywords from the input image header.

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

14.2.1 Parameters

input
[str] full filename with path of input image, an extension name ['sci',1] should be provided if input is a multi-extension FITS file.

14.2.2 Optional Parameters

x
[float, optional] X position from input image

y
[float, optional] Y position from input image

coords
[str, deprecated] [DEPRECATED] full filename with path of file with x,y coordinates Filename given here will be *ignored* if a file has been specified in *coordfile* parameter.

coordfile
[str, optional] full filename with path of file with x,y coordinates

colnames
[str, optional] comma separated list of column names from 'coords' files containing x,y coordinates, respectively. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',... convention.

separator
[str, optional] non-blank separator used as the column delimiter in the coords file

hms
[bool, optional (Default: False)] Produce output in `HH:MM:SS.S` format instead of decimal degrees?

precision
[int, optional] Number of floating-point digits in output values

output
[str, optional] Name of output file with results, if desired

verbose
[bool (Default: False)] Print out full list of transformation results

14.2.3 Returns

ra
[float] Right Ascension of pixel. If more than 1 input value, then it will be a numpy array.

dec
[float] Declination of pixel. If more than 1 input value, then it will be a numpy array.

14.2.4 Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with *AstroDrizzle*. Input images can be updated to use these conventions through the use of the *updatewcs* module the STWCS package.

14.2.5 See Also

`stwcs`

14.2.6 Examples

This task can be run from either the TEAL GUI or from the Python command-line. These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. Run the task using the TEAL GUI under PyRAF:

```
>>> import drizzlepac
>>> epar pixtosky
```

2. Convert a single X,Y position (100,100) from an calibrated ACS image (`j8bt06nyq_flt.fits`) into an undistorted sky position (RA,Dec) without using the TEAL GUI:

```
>>> from drizzlepac import pixtosky
>>> r,d = pixtosky.xy2rd("j8bt06nyq_flt.fits[sci,1]", 100, 100)
```

3. Convert a list of X,Y positions from the file `'xyfile.dat'` for a calibrated ACS image (`j8bt06nyq_flt.fits`) into undistorted sky positions and write out the result to the file 'radec.dat' without using the TEAL GUI:

```
>>> r,d = pixtosky.xy2rd("j8bt06nyq_flt.fits[sci,1]", coordfile="xyfile.dat",
...       output="radec.dat")
```

4. The set of X,Y positions from `'input_flt.fits[sci,1]'` stored as the 3rd and 4th columns from the ASCII file `'xy_sci1.dat'` will be transformed and written out to `'radec_sci1.dat'` using:

```
>>> from drizzlepac import pixtosky
>>> r,d = pixtosky.xy2rd("input_flt.fits[sci,1]", coordfile='xy_sci1.dat',
...                      colnames=['c3','c4'], output="radec_sci1.dat")
```

drizzlepac.pixtosky.**xy2rd**(*input*, *x=None*, *y=None*, *coords=None*, *coordfile=None*, *colnames=None*, *separator=None*, *hms=True*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations from pixel to sky coordinates using STWCS and full distortion models read from the input image header.

14.3 skytopix: Coordinate transformation from sky coordinates

This task allows a user to perform coordinate transformations with the full WCS and distortion model on source positions from sky coordinates to the WCS defined by an image. This task serves as a replacement for the IRAF.STSDAS task *rd2xy*, albeit with the added capability of understanding the full distortion model provided in the headers of images updated for use with astrodrizzle and tweakreg. *skytopix* transforms source positions given as RA, Dec values into pixel positions on the image based on the WCS information and any recognized distortion keywords contained in the input image header.

Authors

Warren Hack

License

http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

14.3.1 Parameters

input

[str] full filename with path of input image, an extension name ['sci',1] should be provided if input is a multi-extension FITS file.

14.3.2 Optional Parameters

ra

[string, optional] RA position in either decimal degrees or HMS format (with or without ':') like '19:10:50.337406303' or '19 10 50.337406303'

dec

[string, optional] Dec position in either decimal degrees or HMS format (with or without ':') like '-60:2:22.186557409' or '-60 2 22.186557409'

coordfile

[str, optional] full filename with path of file with sky coordinates

colnames

[str, optional] comma separated list of column names from 'coordfile' file containing sky coordinates, respectively. Will default to first two columns if None are specified. Column names for ASCII files will use 'c1','c2',... convention.

precision
[int, optional] Number of floating-point digits in output values

output
[str, optional] Name of output file with results, if desired

verbose
[bool] Print out full list of transformation results (default: *False*)

14.3.3 Returns

x
[float] X position of pixel. If more than 1 input value, then it will be a numpy array.

y
[float] Y position of pixel. If more than 1 input value, then it will be a numpy array.

14.3.4 Notes

This task performs a full distortion-correction coordinate transformation based on all WCS keywords and any recognized distortion keywords from the input image header. The transformation recognizes the conventions for describing distortion implemented as part of the SIP and Paper IV conventions used with *AstroDrizzle*. Input images can be updated to use these conventions through the use of the *updatewcs* module the STWCS package.

14.3.5 See Also

stwcs

14.3.6 Examples

This task can be run from either the TEAL GUI, or from the Python command-line. These examples illustrate the syntax that can be used to run the task in a couple of common modes.

1. Run the task using the TEAL GUI under PyRAF:

```
>>> import drizzlepac
>>> epar skytopix
```

2. **Convert a single sky position (0:22:07.0088,-72:03:05.429)**
   from a calibrated ACS image (j94f05bgq_flt.fits) into a pixel position (X,Y) without using the TEAL GUI:

```
>>> from drizzlepac import skytopix
>>> x,y = skytopix.rd2xy("j8bt06nyq_flt.fits[sci,1]",
...                      '0:22:07.0088', '-72:03:05.429')
```

3. **Convert a list of (undistorted) sky positions from the file,**

'radec.dat' for a calibrated ACS image (j8bt06nyq_flt.fits)
into distorted pixel positions, and write out the result to the file 'xypos.dat' without using the TEAL GUI:

```
>>> x,y = skytopix.rd2xy("j8bt06nyq_flt.fits[sci,1]",
...                      coordfile="radec.dat", output="xypos.dat")
```

drizzlepac.skytopix.**rd2xy**(*input*, *ra=None*, *dec=None*, *coordfile=None*, *colnames=None*, *precision=6*, *output=None*, *verbose=True*)

Primary interface to perform coordinate transformations from pixel to sky coordinates using STWCS and full distortion models read from the input image header.

CHAPTER

FIFTEEN

ACS HEADER UPDATE TASK

A task, 'updatenpol', has been written to automate the updating of ACS image headers with the filename of the appropriate NPOLFILE based on the DGEOFILE specified in the image header. This task should be used to update all ACS images prior to processing them with 'astrodrizzle'.

15.1 Updatenpol

updatenpol: Update the header of ACS file(s) with the names of new `NPOLFILE` and `D2IMFILE` reference files for use with the C version of MultiDrizzle (astrodrizzle).

Authors
Warren Hack

License
http://www.stsci.edu/resources/software_hardware/pyraf/LICENSE

Usage
This task can be run from the operating system command line with:

```
updatenpol [options] input [refdir]
```

Command-line Options

input
The specification of the files to be updated, either as a single filename, an ASN table name, or wild-card specification of a list of files.

refdir
The name of the directory containing all the new reference files (`*_npl.fits` and `*_d2i.fits` files). If no directory is given, it will look in *jref$* by default.

`-h`
Print the help (this text).

`-l`
If specified, copy NPOLFILEs and D2IMFILEs to local directory for use with the input files.

-i

If specified, the program will interactively request the exact names of the NPOLFILE and D2IMFILE reference files to be used for updating the header of each file. The value of 'refdir' will be ignored in interactive mode.

Warning: It will ask for the names of the NPOLFILE and D2IMFILE for EACH separate INPUT file when the option *-i* has been specified.

Example

1. This command will update all the FLT files in the current directory with the new NPOLFILE and D2IMFILE reference files found in the 'myjref' directory as defined in the environment:

```
updatenpol *flt.fits myjref$
```

Compatability with MultiDrizzle

The new version of MultiDrizzle (*astrodrizzle) and 'updatewcs* only work with the new NPOLFILE reference file for the DGEO correction (to replace the use of DGEOFILE). In fact, *astrodrizzle* has been extensively modified to prompt the user with a very lengthy explanation on whether it should stop and allow the user to update the header or continue without applying the DGEO correction under circumstances when the NPOLFILE keyword can not be found for ACS.

drizzlepac.updatenpol.**find_d2ifile**(*flist*, *detector*)

Search a list of files for one that matches the detector specified.

drizzlepac.updatenpol.**find_npolfile**(*flist*, *detector*, *filters*)

Search a list of files for one that matches the configuration of detector and filters used.

drizzlepac.updatenpol.**getHelpAsString**(*docstring=False*, *show_ver=True*)

return useful help from a file in the script directory called __taskname__.help

drizzlepac.updatenpol.**help**(*file=None*)

Print out syntax help for running astrodrizzle

Parameters

file : str (Default = None)

If given, write out help to the filename specified by this parameter Any previously existing file with this name will be deleted before writing out the help.

drizzlepac.updatenpol.**run**(*configobj=None*, *editpars=False*)

Teal interface for running this code.

drizzlepac.updatenpol.**update**(*input*, *refdir='jref$'*, *local=None*, *interactive=False*, *wcsupdate=True*)

Updates headers of files given as input to point to the new reference files NPOLFILE and D2IMFILE required with the new C version of MultiDrizzle.

Parameters

input : string or list

Name of input file or files acceptable forms:

- single filename with or without directory
- @-file
- association table
- python list of filenames
- wildcard specification of filenames

refdir : string

Path to directory containing new reference files, either environment variable or full path.

local : boolean

Specifies whether or not to copy new reference files to local directory for use with the input files.

interactive : boolean

Specifies whether or not to interactively ask the user for the exact names of the new reference files instead of automatically searching a directory for them.

updatewcs : boolean

Specifies whether or not to update the WCS information in this file to use the new reference files.

Notes

Warning: This program requires access to the *jref$* directory in order to evaluate the DGEOFILE specified in the input image header. This evaluation allows the program to get the information it needs to identify the correct NPOLFILE.

The use of this program now requires that a directory be set up with all the new NPOLFILE and D2IMFILE reference files for ACS (a single directory for all files for all ACS detectors will be fine, much like jref). Currently, all the files generated by the ACS team has initially been made available at:

```
/grp/hst/acs/lucas/new-npl/
```

The one known limitation to how this program works comes from confusion if more than 1 file could possibly be used as the new reference file. This would only happen when NPOLFILE reference files have been checked into CDBS multiple times, and there are several versions that apply to the same detector/filter combination. However, that can be sorted out later if we get into that situation at all.

Examples

1. A set of associated images specified by an ASN file can be updated to use the NPOLFILEs and D2IMFILE found in the local directory defined using the *myjref$* environment variable under PyRAF using:

```
>>>import updatenpol
>>>updatenpol.update('j8bt06010_asn.fits', 'myref$')
```

2.Another use under Python would be to feed it a specific list of files to be updated using:

```
>>> updatenpol.update(['file1_flt.fits','file2_flt.fits'],'myjref$')
```

3.Files in another directory can also be processed using:

```
>>> updatenpol.update('data$*flt.fits','../new/ref/')
```

CHAPTER

SIXTEEN

REPRODUCING PIPELINE PROCESSING

The task 'runastrodriz' can be used to reproduce the same Drizzle processing that gets performed on HST data when retrieving data from the HST archive.

16.1 Running Astrodrizzle

`runastrodriz` is a module to control operation of astrodrizzle which removes distortion and combines HST images in the pipeline.

16.1.1 Typical Usage

```
>>> runastrodriz.py [-fhibn] inputFilename [newpath]
```

16.1.2 Alternative Usage

```
>>> python
>>> from wfc3tools import runastrodriz
>>> runastrodriz.process(inputFilename,force=False,newpath=None,inmemory=False)
```

16.1.3 GUI Usage under Python

```
>>> python
>>> from stsci.tools import teal
>>> import wfc3tools
>>> cfg = teal.teal('runastrodriz')
```

16.1.4 PyRAF Usage

```
>>> epar runastrodriz
```

16.1.5 Options

If the '-i' option gets specified, no intermediate products will be written out to disk. These products, instead, will be kept in memory. This includes all single drizzle products (*single_sci* and *single_wht*), median image, blot images, and crmask images. The use of this option will therefore require significantly more memory than usual to process the data.

If a value has been provided for the newpath parameter, all processing will be performed in that directory/ramdisk. The steps involved are:

- create a temporary directory under that directory named after the input file
- copy all files related to the input to that new directory
- change to that new directory and run astrodrizzle
- change back to original directory
- move (not copy) ALL files from temp directory to original directory
- delete temp sub-directory

The '-b' option will run this task in BASIC mode without creating headerlets for each input image.

The '-n' option allows the user to specify the number of cores to be used in running AstroDrizzle.

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

d

s

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X