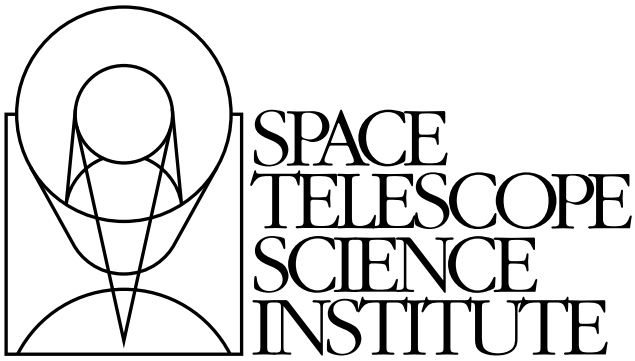

skypac Documentation

Release 0.9 (12-August-2014)

Mihai Cara, Warren Hack, Pey Lian Lim

June 23, 2016

CONTENTS

Contents:

CHAPTER

ONE

SKY MATCHING FOR IMAGE MOSAIC.

A module that provides functions for matching sky in overlapping images.

Authors

Mihai Cara, Warren Hack, Pey-Lian Lim (contact: help@stsci.edu)

License

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

`stsci.skypac.skymatch.`**`TEAL_SkyMatch`**(*input*, *skymethod='globalmin+match'*, *match_down=True*, *skystat='mode'*, *lower=None*, *upper=None*, *nclip=5*, *lsigma=4.0*, *usigma=4.0*, *binwidth=0.1*, *skyuser_kwd='SKYUSER'*, *units_kwd='BUNIT'*, *invsens_kwd=None*, *readonly=True*, *subtractsky=False*, *dq_bits=None*, *optimize='balanced'*, *clobber=False*, *clean=True*, *verbose=True*, *logfile='skymatch.log'*)

TEAL interface for *`skymatch()`*. Most parameters are identical to those of the *`skymatch()`*. Here we mention only the differences:

Parameters

logfile : str (Default = 'skymatch_log.txt')

Store execution log in this file. Always openned in append mode. If not given (`logfile=None`), print to screen instead. NOTE: Unlike *`skymatch()`*, `logfile` can *only* be either a string file name or `None`.

`stsci.skypac.skymatch.`**`skymatch`**(*input*, *skymethod='globalmin+match'*, *skystat='mode'*, *lower=None*, *upper=None*, *nclip=5*, *lsigma=4.0*, *usigma=4.0*, *binwidth=0.1*, *skyuser_kwd='SKYUSER'*, *units_kwd='BUNIT'*, *readonly=True*, *subtractsky=False*, *dq_bits=None*, *optimize='balanced'*, *clobber=False*, *clean=True*, *verbose=True*, *flog='skymatch_log.txt'*)

Standalone task to compute and/or "equalize" sky in input images.

Note: Sky matching ("equalization") is possible only for **overlapping** exposures.

Warning: When `readonly` is `False`, image headers will be modified and image data will be background-subtracted if `subtractsky` is `True`. Remember to back up original copies as desired.

Warning: Unlike previous sky subtraction algorithm used by astrodrizzle, *`skymatch()`* accounts for differences in chip sensitivities by performing sky computations on data multiplied by inverse sensitivity (e.g., value of `PHOTFLAM` in image headers – see "Notes" section below).

Parameters

input : str, list of FileExtMaskInfo

A list of of *FileExtMaskInfo* objects or a string containing one of the following:

- a comma-separated list of valid science image file names (see note below) and (optionally) extension specifications, e.g.: `'j1234567q_flt.fits[1], j1234568q_flt.fits[sci,2]'`;
- 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'`.

Warning: @-file names **MAY NOT** be followed by an extension specification.

Warning: If an association table or a partial file name with wildcard characters is followed by an extension specification, it will be considered that this extension specification applies to **each** file name in the association table or **each** file name obtained after wildcard expansion of the partial file name.

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

Select the algorithm for sky computation:

- **'localmin'**: compute a common sky for all members of *an exposure* (see "Notes" section 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" section 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 com-

puted 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.

---

match_down : bool (Default = True)

Specifies whether the sky *differences* should be subtracted from images with higher sky values (`match_down = True`) to match the image with the lowest sky or sky differences should be added to the images with lower sky values to match the sky of the image with the highest sky value (`match_down = False`).

Note: This setting applies *only* when `skymethod` parameter is either `'match'` or `'globalmin+match'`.

---

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

Statistical method for determining the sky value from the image pixel values. See `computeSky` for more detals.

lower : float, None (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).

upper : float, None (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).

nclip : int (Default = 5)

A non-negative number of clipping iterations to use when computing the sky value.

lsigma : float (Default = 4.0)

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

usigma : float (Default = 4.0)

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

binwidth : float (Default = 0.1)

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

skyuser_kwd : str (Default = 'SKYUSER')

Name of header keyword which records the sky value previously *subtracted* (if `subtractsky` is `True`) from the image data or the *computed* (if `subtractsky` is `False`) sky value. This keyword's value will be updated by *`skymatch()`* (if `readonly` is `False`).

> **Warning:** When `subtractsky` is `True` then `skyuser_kwd` is treated as a **cummulative** value. That is, subtracted sky value will be **added** to the `skyuser_kwd` value and thus `skyuser_kwd` represents *total* sky *subtracted* from the image by the user over the entire "history" of the image. If `skyuser_kwd` is missing in the input image, "previous" sky value will be considered to be 0.0.
> When `subtractsky` is `False` then `skyuser_kwd` represents **computed** sky value and it is **not** treated as a **cummulative** value. Any previous value of the `skyuser_kwd` header keyword will be **overwritten** with the newly computed value.
> Because of different meanings of the value represented by the `skyuser_kwd` header keyword depending on the value of the `subtractsky` parameter, it is important to be consistent and not to mix the two modes when using *`skymatch()`* multiple times on the same images.

units_kwd : str (Default = 'BUNIT')

Name of header keyword which records the units of the data in the image.

invsens_kwd : str, None (Default = '')

Name of header keyword which records the inverse sensitivity of the detector used to acquire data. For `HST` detectors, `'PHOTFLAM` is proportional to detector's inverse sensitivity. It is used to convert electron counts-like to photon counts by multiplying count-like data (or count-rates) by the value indicated by this keyword.

By performing matching using photon counts ("flux units"), one can match images from heterogeneous instruments. Default value `''` or `None` turns off use of inverse sensitivity.

readonly : bool (Default = True)

Report the sky matching values but do not modify the input files.

subtractsky : bool (Default = False)

Subtract computed sky value from image data and add this value to the existing value represented by `skyuser_kwd` (**subtracted sky**) or simply report the computed sky value in the header keyword specified by `skyuser_kwd` (**computed sky**).

> **Warning:** Because `subtractsky` changes the *meaning* of the value of the header keyword `skyuser_kwd` it is important to be consistent in using `subtractsky` parameter: inconsistent use may lead to sky values reported in `skyuser_kwd` header keyword that do not reflect correct sky value *computed for* or *subtracted from* flat-fielded images. A possible workaround is to use different keywords for subtracted and computed sky, keeping in mind that the order of operation will affect reported *computed* sky values.
> Also see warning for `skyuser_kwd` parameter.

Note: When `readonly` is `True`, reported sky values will be consistent with the setting specified by `subtractsky` (as if `readonly` is `False`), however sky values will **NOT** be subtracted from the image data when `subtractsky=True`.

Note: astrodrizzle does not subtract computed sky values from input flat-fielded images. Therefore, when using *`skymatch()`* on images that subsequently will be processed by astrodrizzle it is *recommended* to use the following suggestions:

- If one plans to turn on sky subtraction step in astrodrizzle that will involve additional sky computation (as opposite to using `astrodrizzle`'s `skyuser` or `skyfile` parameters), then it is recommended to set `subtractsky` to `False` and set `skyuser_kwd` to the default value used by `astrodrizzle`: `MDRIZSKY`.
- If one wants to effectively subtract the computed sky values from the flat-fielded image data, then it is recommended to set `subtractsky` to `True`, `skyuser_kwd` parameter to something different from `MDRIZSKY`, (e.g., `SKYUSER`), and set `skyuser` parameter in astrodrizzle to the same value as the values of `skyuser_kwd` used in the call to *`skymatch()`*.

---

dq_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 `dq_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 `dq_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 `dq_bits` to `None` to turn off the use of image's DQ array for sky computations.

In order to reverse the meaning of the `dq_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 `dq_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 `dq_bits` string value of `'~0'` would be equivalent to setting `dq_bits=None`.

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

---

optimize : {'balanced', 'speed', 'inmemory'} (Default = 'balanced')

Specifies whether to optimize execution for speed (maximum memory usage - loaded masks and images are not unloaded until the end of the execution) or use a balanced approach in which a minimal amount of image data is kept in memory and retrieved from disk as needed. The 'inmemory' option is similar to the 'speed' setting except that masks are never saved to the files on a physical disk but are created in memory. The default setting is recommended for most systems.

clobber : bool (Default = False)

When a input image file is in GEIS or WAIVER FITS format it must be converted to simple/MEF FITS file format before it can be used by *skymatch()*. This setting specifies whether any existing simple/MEF files be overwritten during this conversion process. If `clobber=False`, existing simple/MEF FITS files will be opened. If `clobber=True`, input GEIS or WAIVER FITS will be first converted to simple FITS/MEF format overwritting (if necessary) existing files and then these newly created simple FITS/MEF files will be opened.

clean : bool (Default = True)

Specifies whether to delete at the end of the execution any temporary files created by *skymatch()*.

verbose : bool (Default = True)

Specifies whether to print warning messages.

flog : str, file object, MultiFileLog, None (Default = 'skymatch_log.txt')

Log file to which messages shoul be written. It can be a file name, file object, or a MultiFileLog object. The later two allow the log to be written to an existing open output stream passed from the calling function such as astrodrizzle. Log file is always openned in append mode. If not provided (None), print messages to screen only.

Raises

RuntimeError

Could not add an image to mosaic. Possibly this SkyLine does not intersect the mosaic.

TypeError

The `input` argument must be either a Python list of *FileExtMaskInfo* objects, or a string either containing either a comma-separated list file names, or an @-file name.

Notes

skymatch() 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. Values different from zero in user-supplied masks indicate "good" data pixels.

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.

"Surface Brightness":

skymatch() converts "raw" sky values (in image data units) obtained directly from image data to "surface brightness"-like units and all computations are performed in these units. Computed sky surface brightness values are converted back to image data units before being subtracted from the image data and/or reported in the skyuser_kwd in the image header.

This conversion from image data units to "surface brightness"-like units is necessary in order to perform correct sky computations for data from various intsruments/detectors. It accounts for differences in exposure times (if image data are in "counts" units) in each input image, differences in pixel scales of different detector chips (instruments), and detector sensitivities.

For images with data in "counts"-like units, the conversion from data units to surface brightness is given by:

$$sky_{\text{surface brightness}} = sky_{\text{data units}} \cdot \text{PHOTFLAM}/(\text{pixel Area}^2 \cdot \text{EXPTIME})$$

and for image data in "count-rate"-like units, this conversion is given by:

$$sky_{\text{surface brightness}} = sky_{\text{data units}} \cdot \text{PHOTFLAM}/\text{pixel Area}^2.$$

Important Header Keywords:

As discussed above, *skymatch()* uses values of various keywords in image headers to perform conversion of sky values to/from data units from/to surface brightness units. The most important keywords are:

- BUNIT describes the units of the image data. The units of data are determined from the BUNIT header keyword by searching its value for the division sign '/'. If the division sign is not found, then the units are assumed to be "counts". If the division sign is found in the BUNIT value and if the numerator is one of the following: 'ELECTRONS', 'COUNTS', or 'DN', and denumerator is either 'S', 'SEC', or 'SECOND', then the units are assumed to be count-rate.

  If BUNIT is missing then for non-HST images the units will be assumed to be "count-rate", while for HST images (header keyword TELESCOP='HST') the 'INSTRUME' and 'DETECTOR' keywords will be used to infer the units. For the NICMOS instrument, 'UNITCORR' will be used to infer the units. If relevant keywords are missing, the units of image data will be assumed to be "count-rate". Check the log file for selected units.

- EXPTIME – total exposure time, assumed to be in seconds. While the units of EXPTIME are not important for sky computation, it is important that all input images to *skymatch()* use *the same* units. This keyword is used only when inferred units for image data are "count-rates". If EXPTIME is missing when image data units are counts, then variations in exposure time **WILL NOT** be accounted for. First, the primary header of the image file is searched for EXPTIME and if it is not found in the primary header, then image extension is searched for the presense of EXPTIME keyword.

- PHOTFLAM – inverse sensitivity of the detector. At first *skymatch()* will try to detect PHOTFLAM in the image extension header and if not found, it will look for PHOTFLAM in the primary header. If PHOTFLAM is not present at all, the variations in detector sensitivity **WILL NOT** be accounted for.

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 *stsci.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:

- *skymatch()* 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.

@-File Format:

A catalog file containing a science image file and extension specifications and optionally followed by a comma-separated list of mask files and extension specifications (or None).

File names will be stripped of leading and trailing white spaces. If it is essential to keep these spaces, file names may be enclosed in single or double quotation marks. Quotation marks may also be required when file names contain special characters used to separate file names and extension specifications: ,[]{}

Extension specifications must follow the file name and must be delimited by either square or curly brackets. Curly brackets allow specifying multiple comma-separated extensions: integer extension numbers and/or tuples ('ext name', ext version).

Some possible ways of specifying extensions:

[1] – extension number

['sci',2] – extension name and version

{1,4,('sci',3)} – multiple extension specifications, including tuples

{('sci',*)} – wildcard extension versions (i.e., all extensions with extension name 'sci')

['sci'] – equivalent to ['sci',1]

{'sci'} – equivalent to {('sci',*)}

For extensions in the science image for which no mask file is provided, the corresponding mask file names may be omitted (but a comma must still be used to show that no mask is provided in that position) or None can be used in place of the file name. NOTE: 'None' (in quotation marks) will be interpreted as a file named None.

Some examples of possible user input:

image1.fits{1,2,('sci',3)},mask1.fits,,mask3.fits[0]

In this case:

`image1.fits`[1] is associated with `mask1.fits`[0];

`image1.fits`[2] does not have an associated mask;

`image1.fits`['sci',3] is associated with `mask3.fits`[0].

– Assume `image2.fits` has 4 'SCI' extensions:

image2.fits{'sci'},None,,mask3.fits

In this case:

`image2.fits`['sci',1] and `image2.fits`['sci',2] **and** `image2.fits`['sci',4] do not have an associated mask;

`image2.fits`['sci',3] is associated with `mask3.fits`[0]

Note: User mask data that indicate what pixels in the input `image` should be used for sky computations (`1`) and which pixels should **not** be used for sky computations (`0`).

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, *`skymatch()`* 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. Values different from zero in user-supplied masks indicate "good" data pixels. 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.

Examples

1. This task can be used to match skies of a set of ACS images simply with:

```
>>> from stsci.skypac import skymatch
>>> skymatch.skymatch('j*q_flt.fits')
```

2. The TEAL GUI can be used to run this task using:

```
>>> from stsci.skypac import skymatch
>>> epar skymatch
```

or from a general Python command line:

```
>>> from stsci.skypac import skymatch
>>> from stsci.tools import teal
>>> teal.teal('skymatch')
```

CHAPTER

TWO

SKYLINE (CHIP OUTLINE ON THE SKY) MANAGEMENT FOR IMAGE MOSAIC.

This module provides support for working with footprints on the sky. Primary use case would use the following generalized steps:

1. Initialize *SkyLine* objects for each input image. This object would be the union of all the input image's individual chips WCS footprints.
2. Determine overlap between all images. The determination would employ a recursive operation to return the extended list of all overlap values computed as [img1 vs [img2,img3,...,imgN],img2 vs [img3,...,imgN],...]
3. Select the pair with the largest overlap, or the pair which produces the largest overlap with the first input image. This defines the initial reference *SkyLine* object.
4. Perform some operation on the 2 images: for example, match sky in intersecting regions, or aligning second image with the first (reference) image.
5. Update the second image, either apply the sky value or correct the WCS, then generate a new *SkyLine* object for that image.
6. Create a new reference *SkyLine* object as the union of the initial reference object and the newly updated *SkyLine* object.
7. Repeat Steps 2-6 for all remaining input images.

This process will work reasonably fast as most operations are performed using the *SkyLine* objects and WCS information solely, not image data itself.

Authors
Mihai Cara, Warren Hack, Pey-Lian Lim (contact: help@stsci.edu)

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

class stsci.skypac.skyline.**SkyLineMember**(*image*, *ext*, *dq_bits=0*, *dqimage=None*, *dqext=None*, *usermask=None*, *usermask_ext=None*)

Container for *SkyLine* members that holds information about properties of a *single* extension (chip) in a FITS image such as:

- WCS of the chip image;
- bounding spherical polygon;
- file name and extension from which the chip's image has originated;
- information required for unit conversions (EXPTIME, PHOTFLAM, BUNIT, etc.);
- user mask and DQ array associated with chip's image data.

Parameters

image : ImageRef

An *ImageRef* object that refers to an open FITS file

ext : tuple, int, str

Extension specification in the `image` the *SkyLineMember* object will be associated with.

An int `ext` specifies extension number. A tuple in the form (str, int) specifies extension name and number. A string `ext` specifies extension name and the extension version is assumed to be 1. See documentation for `astropy.io.fits.getData` for examples.

dq_bits : int, 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 `dq_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 `dq_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 by the `usermask` parameter.

---

dqimage : ImageRef

An *ImageRef* object that refers to an open FITS file that has DQ data of the input `image`.

Note: When DQ data are located in the same FITS file as the science image data (e.g., HST/ACS, HST/WFC3, etc.), `dqimage` may point to the same *ImageRef* object. In this case the reference count of the *ImageRef* object must be increased adequately.

---

dqext : tuple, int, str

Extension specification of the `dqimage` that contains `image`'s DQ information. See help for `ext` for more details on acceptable formats for this parameter.

usermask : ImageRef

An *ImageRef* object that refers to an open FITS file that has user mask data that indicate what pixels in the input `image` should be used for sky computations (`1`) and which pixels should **not** be used for sky computations (`0`).

usermask_ext : tuple, int, str

Extension specification of the `usermask` mask file that contains user's mask data that should be associated with the input `image` and `ext`. See help for `ext` for more details on acceptable formats for this parameter.

class `stsci.skypac.skyline.`**`SkyLine`**(*mlist*)

Manage outlines on the sky.

Skylines are designed to capture and manipulate HST WCS image information as spherical polygons. They are represented by the `SkyLine` class, which is an extension of `SphericalPolygon` class.

Each skyline has a list of members, `members`, and a composite spherical polygon, `polygon`, members. The polygon has all the functionalities of `SphericalPolygon`.

Each *`SkyLine`* has a list of *`members`* and a composite *`polygon`* with all the functionalities of `SphericalPolygon`.

Each member in `members` belongs to the `SkyLineMember` class, which contains image name (with path if given), science extension(s), and composite WCS and polygon of the extension(s). All skylines start out with a single member from a single image. When operations are used to find composite or intersecting skylines, the resulting skyline can have multiple members.

For example, a skyline from an ACS/WFC full-frame image would give 1 member, which is a composite of extensions 1 and 4. A skyline from the union of 2 such images would have 2 members, and so forth.

Parameters

fname : str

FITS image. `None` to create empty *`SkyLine`*.

ext : a list of tuples ('extname',extver).

`add_image`(*other*)

Return a new *`SkyLine`* that is the union of *self* and *other*.

> **Warning:** `SkyLine.union` only returns *`polygon`* without *`members`*.

Parameters

other : *`SkyLine`* object

Examples

```
>>> s1 = SkyLine('image1.fits')
>>> s2 = SkyLine('image2.fits')
>>> s3 = s1.add_image(s2)
```

`find_intersection`(*other*)

Return a new *`SkyLine`* that is the intersection of *self* and *other*.

> **Warning:** `SkyLine.intersection` only returns *`polygon`* without *`members`*.

Parameters

other : *`SkyLine`* object

Examples

```
>>> s1 = SkyLine('image1.fits')
>>> s2 = SkyLine('image2.fits')
>>> s3 = s1.find_intersection(s2)
```

`find_max_overlap`(*skylines*)

Find *`SkyLine`* from a list of *skylines* that overlaps the most with *self*.

Parameters
skylines : list

A list of *SkyLine* instances.

Returns
max_skyline : *SkyLine* instance or None

SkyLine that overlaps the most or None if no overlap found. This is *not* a copy.

max_overlap_area : float

Area of intersection.

static max_overlap_pair(*skylines*)
Find a pair of skylines with maximum overlap.

Parameters
skylines : list

A list of *SkyLine* instances.

Returns
max_pair : tuple

Pair of *SkyLine* objects with max overlap among given *skylines*. If no overlap found, return None. These are *not* copies.

to_wcs()
Combine HSTWCS objects from all *members* and return a new HSTWCS object. If no *members*, return None.

Warning: This cannot return WCS of intersection.

is_mf_mosaic
returns *True* if *SkyLine* members are from distinct image files (multi-file mosaic) and *False* otherwise.

members
List of *SkyLineMember* objects that belong to *self*. Duplicate members are discarded. Members are kept in the order of their additions to *self*.

polygon
SphericalPolygon portion of *SkyLine* that contains the composite skyline from *members* belonging to *self*.

CHAPTER

THREE

UTILITY FUNCTIONS FOR PARSING USER CATALOG FILES FOR SKYPAC.

Module for parsing `@-files` or user input strings for use by `stsci.skypac` module.

Authors
Mihai Cara (contact: help@stsci.edu)

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

class `stsci.skypac.parseat.`**`FileExtMaskInfo`**(*default_ext=('SCI', '*')*, *default_mask_ext=0*, *clobber=False*, *doNotOpenDQ=False*, *fnamesOnly=False*, *im_fmode='update'*, *dq_fmode='readonly'*, *msk_fmode='readonly'*)

A class that holds image, dq, user masks, and extensions to be used with these files. It is designed to facilitate keeping track of user input in catalog files.

This class is intended to be used primarily for functions such as *parse_at_line()* and other related functions as a return value. It is also used to initialize `skypac.skyline.SkyLine` objects.

FileExtMaskInfo was designed to be used in a specific ordered workflow. Below is a typical use of this class:

1. Initialize the object with the desired settings for default extensions to be used with the files (when a specific extension for a file is not provided) and the open modes for the files;
2. Add image file using *image()*;
3. Add image's extension(s) using *append_ext()*;
4. [Optional; can be performed at any **later** stage] Add DQ file and extensions using *DQimage()* and *dqext()* methods;
5. Append mask files and extensions using *append_mask()*;
6. [Optional] Finalize the *FileExtMaskInfo()* object.

Parameters

default_ext : int, tuple (Default = ('SCI','*'))

Default extension to be used with image files that to not have an extension specified.

default_mask_ext : int, tuple (Default = 0)

Default extension to be used with image mask files that to not have an extension specified.

clobber : bool (Default = False)

If a file being appended is in GEIS or WAIVER FITS format, should any existing MEF files be overwritten?

doNotOpenDQ : bool (Default = False)

Should the DQ files be oppened when simultaneously with the image files?

fnamesOnly : bool (Default = False)

Return file names only, or open the files and return `ImageRef` objects?

im_fmode : str (Default = 'update')

File mode to be used to open image FITS file. See `astropy.io.fits.open` for more details.

dq_fmode : str (Default = 'readonly')

File mode to be used to open DQ FITS file. This is valid only if the DQ model of the image file is 'external' (see documentation for `ImageRef` for more details). For 'intrinsic' DQ model the DQ files will use the same setting as for `im_fmode`.

msk_fmode : str (Default = 'readonly')

File mode to be used to open mask files.

Attributes

clobber	(bool) If a file being appended is in GEIS or WAIVER FITS format, should any existing MEF files be overwritten?
dq_bits	(int) Bitmask specifying what pixels in the mask should be removed (or kept) with the precise interpretation being left to the user. This flag is not used by this class but was designed to be passed to other functions that will use *`FileExtMaskInfo`*.

append_ext(*ext*)

Append extensions to the list of "selected" extensions for the image file.

Note: This function *appends* the extensions. If it is desired to *set* the extensions, use `replace_ext()` instead.

Parameters

ext : int, tuple, None, list

Extension specification: None, an integer extension *number*, a tuple (*extension number*, *extension version*) where extension version can be '*' which will be replaced with the extension versions of all extensions having given extension name. If ext is None, it will be replaced with the default extension specification set during the initialization of the *`FileExtMaskInfo`* object.

append_mask(*mask*, *ext*, *mask_stat=None*)

Append a mask image and its extension(s).

Note: Mask files and extensions are kept in ordered lists and their order is significant: the first mask file-extension pair is associated with the first extension of the science image file set with *`append_ext()`* and so on.

Parameters

mask : str, ImageRef, None

Mask image file. Can be a string file name, an `ImageRef` object (*only* if *`fnamesOnly`*=`False`), or None (to act as a place holder in the ordered list of extensions).

ext : int, tuple, None, list

Extension specification: None, an integer extension *number*, a tuple (*extension number*, *extension version*) where extension version can be '*' which will be replaced with the extension versions of all extensions having given extension name. If ext is None, it will be replaced with the default extension specification for mask images set during the initialization of the *`FileExtMaskInfo`* object.

mask_stat : `os.stat_result` (Default = None)

An `os.stat_result` structure for the input `mask` file. If `None`, then *`append_mask()`* will compute `stat` for the input `mask` file.

Raises

RuntimeError

Raised if attempting to add masks when the science image was not yet set.

AssertionError

Raised if *`finalized`*=`True`.

TypeError

Raised if `mask` is an `ImageRef` object but *`fnamesOnly`*=`True`or if `mask` argument is of incorrect type.

ValueError

If `mask` is an `ImageRef`, it must *not* be closed.

`clear_masks`()

Remove all attached mask files and extensions.

`convert2ImageRef`()

Replace any existing file names with opened `ImageRef` objects and change the *`fnamesOnly`* property to `False`.

Note: The *`finalized`* property will not be modified.

Warning: The *`FileExtMaskInfo`* must not have been finalized (*`finalized`*=`False`) and must contain file names only (*`fnamesOnly`*=`True`).

Raises

AssertionError

Raised if *`finalized`*=`True` or *`fnamesOnly`*=`False`.

See also:

`release_all_images`

`finalize`(*toImageRef=False*)

Finalize the object by trimming or extending mask image lists to match the number of science image extensions.

In principle, the number of mask files and their extensions need not be equal to the number of extensions specified for the science image. If the number of masks/extensions is smaller than the number of science extensions, the list of mask extensions will be appended with None (if *fnamesOnly*=True) or dummy ImageRef (if *fnamesOnly*= False) until the number of mask extensions is equal to the number of science image extensions. If the number of mask extensions is larger than the number of science image extensions, the list of mask extensions will be trimmed to match the number of science image extensions. The trimmed out mask files (if represented by ImageRef) will be "released".

info()
Print information about the state of the object.

release_all_images()
Release all images if *fnamesOnly* is False and replace any existing ImageRef with their *original* file names.

Note: This will set the *fnamesOnly* property to True and the *finalized* property to False.

See also:

convert2ImageRef

replace_fext(*ext*)
Replace/set image file extension list.

See also:

append_ext

DQimage
DQ image (file or ImageRef object depending on the *fnamesOnly* value).

Getter
Get the ImageRef DQ image object.

Setter
Set the DQ file.

Type
str, ImageRef

count
Number of extensions associated with the image file.

dqext
FITS extensions associated with the DQ file.

fext
FITS extensions associated with the image file.

finalized
Is the *FileExtMaskInfo* object finalized?

fnamesOnly
Was the *FileExtMaskInfo* initialized to return file names or the ImageRef objects?

image
Image file name or the associated ImageRef object (depending on the *fnamesOnly* value).

Getter
Get the ImageRef image object.

Setter
Set the image file.

Type
str, ImageRef, None

Note: Setting the image will re-initialize *FileExtMaskInfo*. All previous settings will be lost and previously attached files will be released/deleted.

---

mask_images
Mask image file names or ImageRef object depending on the *fnamesOnly* value.

maskext
FITS extensions associated with the mask files.

stsci.skypac.parseat.**parse_at_file**(*fname*, *default_ext=('SCI', '*')*, *default_mask_ext=0*, *clobber=False*, *fnamesOnly=False*, *doNotOpenDQ=False*, *match2Images=None*, *im_fmode='update'*, *dq_fmode='readonly'*, *msk_fmode='readonly'*, *logfile=None*, *verbose=False*)

This function is similar to *parse_at_line()*, the main difference being that is can parse multiple (EOL terminated) lines of the format described in the documentation for *parse_at_line()*.

Below we describe only differences bewtween this function and *parse_at_line()*.

Parameters
fname : str

File name of the catalog file.

match2Images : list of str, list of ImageRef, None (Default = None)

List of file names or ImageRef objects whose mask specifications are to be read from the catalog file. Mask specifications for other files in the catalog that do not match the files in the `match2Images` list will be ignored. If `match2Images` is `None`, then all files from the catalog will be read.

logfile : str, file, MultiFileLog, None (Default = None)

Specifies the log file to which the messages should be printed. It can be a file name, a file object, a MultiFileLog object, or None.

Returns
list

Returns a list of filenames if `fnamesOnly=True` or a list of *FileExtMaskInfo* objects if `fnamesOnly=False`.

stsci.skypac.parseat.**parse_at_line**(*fstring*, *default_ext=('SCI', '*')*, *default_mask_ext=0*, *clobber=False*, *fnamesOnly=False*, *doNotOpenDQ=False*, *im_fmode='update'*, *dq_fmode='readonly'*, *msk_fmode='readonly'*, *verbose=False*)

Parse a line from a catalog file containing a science image file and extension specifications and optionally followed by a comma-separated list of mask files and extension specifications (or None).

File names will be stripped of leading and trailing white spaces. If it is essential to keep these spaces, file names may be enclosed in single or double quotation marks. Quotation marks may also be required when file names contain special characters used to separate file names and extension specifications: ,[]{}

Extension specifications must follow the file name and must be delimited by either square or curly brackets. Curly brackets allow specifying multiple comma-separated extensions: integer extension numbers and/or tuples ('ext name', ext version).

Some possible ways of specifying extensions:

[1] – extension number

['sci',2] – extension name and version

{1,4,('sci',3)} – multiple extension specifications, including tuples

{('sci',*)} – wildcard extension versions (i.e., all extensions with extension name 'sci')

['sci'] – equivalent to ['sci',1]

{'sci'} – equivalent to {('sci',*)}

For extensions in the science image for which no mask file is provided, the corresponding mask file names may be omitted (but a comma must still be used to show that no mask is provided in that position) or None can be used in place of the file name. NOTE: 'None' (in quotation marks) will be interpreted as a file named None.

Some examples of possible user input:

image1.fits{1,2,('sci',3)},mask1.fits,,mask3.fits[0]

In this case:

`image1.fits`[1] is associated with `mask1.fits`[0];

`image1.fits`[2] does not have an associated mask;

`image1.fits`['sci',3] is associated with `mask3.fits`[0].

– Assume `image2.fits` has 4 'SCI' extensions:

image2.fits{'sci'},None,,mask3.fits

In this case:

`image2.fits`['sci',1] and `image2.fits`['sci',2] **and** `image2.fits`['sci',4] do not have an associated mask;

`image2.fits`['sci',3] is associated with `mask3.fits`[0]

Note: Wildcard extension version in extension specification can be expanded *only* when `fnamesOnly=False`.

Parameters

fstring : str

A comma-separated string describing the image file name and (optionally) followed by the extension specifier (e.g., [sci,1,2], or [sci]). The image file name may be followed (comma-separated) by optional mask file names (and their extension specifier).

File and extension names may NOT contain leading and/or trailing spaces, commas, and/or square or curly brakets.

default_ext : int, tuple (Default = ('SCI','*'))

Default extension to be used with image files that to not have an extension specified.

default_mask_ext : int, tuple (Default = 0)

Default extension to be used with image mask files that to not have an extension specified.

fnamesOnly : bool (Default = False)

Return file names only, or open the files and return `ImageRef` objects?

doNotOpenDQ : bool (Default = False)

Should the DQ files be oppened when simultaneously with the image files?

im_fmode : str (Default = 'update')

File mode to be used to open image FITS file. See `astropy.io.fits.open` for more details.

dq_fmode : str (Default = 'readonly')

File mode to be used to open DQ FITS file. This is valid only if the DQ model of the image file is 'external' (see documentation for `ImageRef` for more details). For 'intrinsic' DQ model the DQ files will use the same setting as for `im_fmode`.

msk_fmode : str (Default = 'readonly')

File mode to be used to open mask files.

verbose : bool (Default = True)

Specifies whether to print warning messages.

Returns

FileExtMaskInfo

A *FileExtMaskInfo* object.

Raises

ValueError

- Input argument 'fstring' must be a Python string.
- Input argument 'fstring' contains either unbalanced or nested square brackets.
- Extension specification must be preceeded by a valid image file name.

`stsci.skypac.parseat.`**`parse_cs_line`**(*csline*, *default_ext=('SCI', '*')*, *clobber=False*, *fnamesOnly=False*, *doNotOpenDQ=False*, *im_fmode='update'*, *dq_fmode='readonly'*, *msk_fmode='readonly'*, *logfile=None*, *verbose=False*)

This function is similar to *parse_at_line()*, the main difference being the content of the input string: a list of comma-separated *science* image file names. No masks can be specified and file names must be valid (i.e., None is not allowed). Extension specifications are allowed and must folow the same sintax as described for *parse_at_line()*.

Below we describe only differences bewtween this function and *parse_at_line()*.

Parameters

csline : str

User input string that needs to be parsed containing one of the following:

- a comma-separated list of valid science image file names (see note below) and (optionally) extension specifications, e.g.: `'j1234567q_flt.fits[1], j1234568q_flt.fits[sci,2]'`;
- an @-file name, e.g., `'@files_to_match.txt'`.

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'`.

Warning: @-file names **MAY NOT** be followed by an extension specification.

Warning: If an association table or a partial file name with wildcard characters is followed by an extension specification, it will be considered that this extension specification applies to **each** file name in the association table or **each** file name obtained after wildcard expansion of the partial file name.

logfile : str, file, MultiFileLog, None (Default = None)

Specifies the log file to which the messages should be printed. It can be a file name, a file object, a MultiFileLog object, or None.

Returns

list

Returns a list of filenames if `fnamesOnly=True` or a list of *FileExtMaskInfo* objects if `fnamesOnly=False`.

CHAPTER

FOUR

POLYGON FILLING ALGORITHM

Polygon filling algorithm.

Authors
Nadezhda Dencheva, Mihai Cara (contact: help@stsci.edu)

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

class stsci.skypac.region.**Region**(*rid*, *coordinate_system*)
Base class for regions.

Parameters
rid : int or string

region ID

coordinate_system : astropy.wcs.CoordinateSystem instance or a string

in the context of WCS this would be an instance of wcs.CoordinateSysem

scan(*mask*)
Sets mask values to region id for all pixels within the region. Subclasses must define this method.

Parameters
mask : ndarray

a byte array with the shape of the observation to be used as a mask

Returns
mask : array where the value of the elements is the region ID or 0 (for

pixels which are not included in any region).

class stsci.skypac.region.**Edge**(*name=None*, *start=None*, *stop=None*, *next=None*)
Edge representation

An edge has a "start" and "stop" (x,y) vertices and an entry in the GET table of a polygon. The GET entry is a list of these values:

[ymax, x_at_ymin, delta_x/delta_y]

compute_AET_entry(*edge*)
Compute the entry for an edge in the current Active Edge Table

[ymax, x_intersect, 1/m] note: currently 1/m is not used

compute_GET_entry()
Compute the entry in the Global Edge Table

[ymax, x@ymin, 1/m]

class stsci.skypac.region.**Polygon**(*rid*, *vertices*, *coord_system='Cartesian'*)

Represents a 2D polygon region with multiple vertices

Parameters

rid : string

polygon id

vertices : list of (x,y) tuples or lists

The list is ordered in such a way that when traversed in a counterclockwise direction, the enclosed area is the polygon. The last vertex must coincide with the first vertex, minimum 4 vertices are needed to define a triangle

coord_system : string

coordinate system

get_edges()

Create a list of Edge objects from vertices

scan(*data*)

This is the main function which scans the polygon and creates the mask

Parameters

data : array

the mask array it has all zeros initially, elements within a region are set to the region's ID

Algorithm:

- Set the Global Edge Table (GET)

- Set y to be the smallest y coordinate that has an entry in GET

- Initialize the Active Edge Table (AET) to be empty

- For each scan line:

1.Add edges from GET to AET for which ymin==y

2.Remove edges from AET fro which ymax==y

3.Compute the intersection of the current scan line with all edges in the AET

4.Sort on X of intersection point

5.Set elements between pairs of X in the AET to the Edge's ID

update_AET(*y*, *AET*)

Update the Active Edge Table (AET)

Add edges from GET to AET for which ymin of the edge is equal to the y of the scan line. Remove edges from AET for which ymax of the edge is equal to y of the scan line.

CHAPTER

FIVE

SKY STATISTICS FUNCTIONS FOR SKYPAC.

Sky statistics computation class for `skymatch` and `_weighted_sky`.

Authors
Mihai Cara (contact: help@stsci.edu)

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

class `stsci.skypac.skystatistics.`**`SkyStats`**(*skystat='mean'*, ***kwargs*)
This is a superclass build on top of `stsci.imagestats.ImageStats`. Compared to ImageStats, SkyStats has "persistent settings" in the sense that object's parameters need to be set once and these settings will be applied to all subsequent computations on different data.

Initializes the SkyStats object.

Parameters
skystat : str

Sets the statistics that will be returned by the *`calc_sky`*. The following statistics are supported: 'mean', 'mode', 'midpt', and 'median'. First three statistics have the same meaning as in stsdas.toolbox.imgtools.gstatistics while `skystat`='median' will compute the median of the distribution.

kwargs : dict

A dictionary of optional arguments to be passed to `ImageStats`.

`calc_sky`(*data*)
Computes statistics on data.

Parameters
data : numpy.ndarray

A numpy array of values for which the statistics needs to be computed.

Returns
statistics : tuple

A tuple of two values: (`skyvalue`, `npix`), where `skyvalue` is the statistics specified by the `skystat` parameter during the initialization of the *`SkyStats`* object and `npix` is the number of pixels used in comuting the statistics reported in `skyvalue`.

class `stsci.skypac.skystatistics.`**`SkyStats`**(*skystat='mean'*, ***kwargs*)
This is a superclass build on top of `stsci.imagestats.ImageStats`. Compared to ImageStats, SkyStats has "persistent settings" in the sense that object's parameters need to be set once and these settings will be applied to all subsequent computations on different data.

Initializes the SkyStats object.

Parameters

skystat : str

Sets the statistics that will be returned by the *calc_sky*. The following statistics are supported: 'mean', 'mode', 'midpt', and 'median'. First three statistics have the same meaning as in stsdas.toolbox.imgtools.gstatistics while `skystat`='median' will compute the median of the distribution.

kwargs : dict

A dictionary of optional arguments to be passed to `ImageStats`.

calc_sky(*data*)

Computes statistics on data.

Parameters

data : numpy.ndarray

A numpy array of values for which the statistics needs to be computed.

Returns

statistics : tuple

A tuple of two values: (`skyvalue`, `npix`), where `skyvalue` is the statistics specified by the `skystat` parameter during the initialization of the *SkyStats* object and `npix` is the number of pixels used in comuting the statistics reported in `skyvalue`.

CHAPTER

SIX

UTILITY FUNCTIONS FOR SKYPAC.

This module provides utility functions for use by `stsci.skypac` module.

Authors
Mihai Cara (contact: help@stsci.edu)

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

class `stsci.skypac.utils.`**`MultiFileLog`**(*console=True*, *enableBold=True*, *flog=None*, *append=True*, *autoflush=True*, *appendEOL=True*)

This is a class that facilitates writting to multiple files. *`MultiFileLog`* stores multiple file objects and can write the same log entry to all of them. It also facilitates controlling when a function can close a log file. Finally, it provides some utility functions that automate such things as appending EOL at the end of the log entry, flushing the files (to avoid losing log entries in case of uncaught exceptions), displaying `WARNING`, `ERROR`, etc. in bold on standard streams (sys.stdout, etc.)

Parameters

console : bool (Default = True)

Enables writting to the standard output.

enableBold : bool (Default = True)

Enable or disable writing bold text to console, e.g., 'WARNING:' 'ERROR:', etc.

flog : str, file, None, list of str or file objects (Default = None)

File name or file object to be added to *`MultiFileLog`* during the initialization. More files can be added lated with *`add_logfile()`*.

append : bool (Default = True)

Default open mode for the files that need to be opened (e.g., that are passed as file names). If `append` is `True`, new files added to a *`MultiFileLog`* object will be opened in the "append" mode: new log entries will be appended to existing files – same as mode 'a' of standard function `open()`. When `append` is `False`, any existing files will be overwritten.

autoflush : bool (Default = True)

Indicates whether or not to flush files after each log entry.

appendEOL : bool (Default = True)

Indicates whether or not to add EOL at the end of each log entry.

`add_logfile`(*flog*, *initial_skip=2*, *can_close=None*, *mode=None*)

Add (and open, if necessary) a log file to the *`MultiFileLog`* object.

Parameters

flog : str, file, None, list of str or file objects (Default = None)

File name or file object to be added.

initial_skip : int (Default = 2)

The number of blank line to be written to the file *if not* at the beginning of the file.

can_close : bool, None (Default = None)

Indicates whether the file object can be closed by the `close()` function:

- `can_close=True` – file can be closed by the `close()` function;
- `can_close=False` – file will not be closed by the `close()` function;
- `can_close=None` – automatic selection based on the type of the `flog` argument: `True` if flog is a string (e.g., file name), `False` otherwise.

mode : str, None (Default = None)

File open mode: same meaning as the `mode` parameter of the Python's built-in `open()` function. If `None`, the mode will be inherited from file open mode set during initialization.

Returns

flog : file object

File object of newly opened (or attached) file.

close()

Close all files *opened by* `MultiFileLog`.

It will close all files *opened by* `MultiFileLog` - essentially, all files added as file name. It will *not* close files added as file handles.

enable_console(*enable=True*, *enableBold=True*)

Enable output to the standard console – `sys.stdout`.

Parameters

enable : bool (Default = True)

Enable or disable output to `sys.stdout`.

enableBold : bool (Default = True)

Enable or disable writing bold text to console, e.g., 'WARNING:' 'ERROR:', etc.

error(*msg*, **fmt*)

Prints an error message. The word 'ERROR:' will be printed as bold on the console and enclosed in asterisks (*) in a disk file.

flush()

Flush all files.

important(*msg*, **fmt*)

Prints an "important" message. The word 'IMPORTANT:' will be printed as bold on the console and enclosed in asterisks (*) in a disk file.

logentry(*msg*, **fmt*, ***skip*)

Write a log entry to the log files.

Parameters

msg : str

String to be printed. Can contain replacement fields delimited by braces {}.

fmt

Parameters to be passed to `str.format()` method for formatting the string `msg`.

skip : int

Number of empty lines that should follow the log message.

Examples

```
>>> logentry('Sky background for chip {} is {}', 'SCI1', 10.0, skip=2)
will print:
Sky background for chip SCI1 is 10.0
followed by two blank lines.
```

print_endlog_msg()
Print a message ("Log written to...") indicating the end of the log.

set_close_flag(*fh*, *can_close=True*)
Modify the "can be closed" status of a given file.

Parameters
fh : file object

can_close : bool (Default = True)

Indicates if the file can be closed by the *close()* function.

Note: File object `fh` **must** have been already added to the *MultiFileLog* object.

---

skip(*nlines=1*)
Skip a specified number of blank lines.

Parameters
nlines : int (Default = 1)

unclose_copy()
Return a copy of the *MultiFileLog* object with all the attached files marked as "keep open", that is, the *close()* will not close these files.

This is useful before passing the *MultiFileLog* object to a function that may add its own log files, and then attempt to close the files with the *close()* method. Thus, by passing an "unclose copy", one can be sure that the files opened at the top level will not be closed by other functions to which the *MultiFileLog* object may be passed.

warning(*msg*, **fmt*)
Prints a warning message. The word 'WARNING:' will be printed as bold on the console and enclosed in asterisks (*) in a disk file.

count
Return the number of files attached to the *MultiFileLog* object excluding the `sys.stdout` file.

class stsci.skypac.utils.**ResourceRefCount**(*resource*, *resource_close_fnct=None*, **close_args*, ***close_kwargs*)
A class that implements reference counting for various resources: file objects, PyFITS HDU lists, etc. It is intended to be used as a mechanism of controlling the "lifespan" of resources that can be used in different parts of the code "indipendently". The resource is kept open as long as the reference count is larger than zero. Once the reference count is decreased to 0, the resource is automatically closed.

Note: The reference count of the newly created *ResourceRefCount* object is set to 0. It is user's responsibility to increment the reference count of this object through the call to *hold()* function.

Parameters

resource

An object who must be kept open or closed based on reference count.

resource_close_fnct : function (Default = None)

The function (usually a method of the attached resource), that can "close" the resource.

close_args : tuple

Positional arguments to be passed to the `resource_close_fnct()` function.

close_kwargs : dict

Keyword arguments to be passed to the `resource_close_fnct()` function.

hold()
Increment the reference count of the attached resource.

is_subscribed_on_close(*obj*)
Check if the object is subscribed to on close notifications.

release()
Decrement reference count of the attached resource. If the reference count reaches zero, call all registered "on close" notify callbacks, and then call the "close" method on the resource which was set at initialization of the *ResourceRefCount* object. Finally, the *resource* property of the *ResourceRefCount* object will be set to `None`.

subscribe_close_notify(*obj*, *callback=None*)
Set the object and its method that need to be called when the resource is about to be closed.

unsubscribe_close_notify(*obj*)
Remove the object (and its method) from the list of callbacks that need to be notified of impending closing of the resource.

closed
Indicates if the resource is "closed".

refcount
Reference count.

resource
Resource attached to a *ResourceRefCount* object.

class `stsci.skypac.utils.`**ImageRef**(*hdulist_refcnt=None*)
A lightweight class that supports reference counting for FITS images and holds a *ResourceRefCount* object. It provides several attributes that describe main characteristics of the image (file) and essential functions for manipulating reference count.

Parameters

hdulist_refcnt : ResourceRefCount

A *ResourceRefCount* object holding a `astropy.io.fits.HDUList` object.

Attributes

filename	(str) Name of the opened file. Can be `None` for in-memory created `astropy.io.fits.HDUList` objects.
can_reload_data	(bool) `True` for files attached to a physical file, `False` for in-memory `astropy.io.fits.HDUList` objects.
original_fname	(str) Original name of the file as requested by the user. *Note:* may be different from `filename` if the orininal file was in GEIS or WAIVER FITS format and subsequently was converted to a MEF FITS format. In that case this attribute will show the name of the original GEIS or WAIVER FITS file.
original_ftype	(str) Type of the original file. Can take one of the following values: 'MEF', 'SIMPLE', 'GEIS', 'WAIVER', or 'UNKNOWN'.
original_exists	(bool) Indicates if the physical file exists. It is `False` for in-memory images.
mef_fname	(str, None) Name of the MEF FITS file if exists, `None` otherwise.
mef_exists	(bool) Indicates whether the MEF FITS file exists.
DQ_model	(str, None) Type of the DQ model: 'external' for WFPC, WFPC2, and FOC instruments (or non-HST data if set so) that have DQ data in a separate (from image) file and 'intrinsic' for ACS, etc. images that have DQ extensions in the image file. It is `None` if the image does not have DQ data.
telescope	(str) Telescope that acquired the image.
instrument	(str) Instrument used to acquire data.
fmode	(str) File mode used to open FITS file. See `astropy.io.fits.open` for more details.
memmap	(bool) Is the `astropy.io.fits.HDUList` memory mapped?

hold()

Increment the reference count of the attached resource.

info(*self*, *fh=sys.stdout*)

Print a summary of the object attributes.

release()

Decrement reference count of the attached resource. If the reference count reaches zero, the attached *ResourceRefCount* object will be closed and set to `None`.

set_HDUList_RefCount(*hdulist_refcnt=None*)

Set (attach) a new *ResourceRefCount* object that holds a `astropy.io.fits.HDUList` object. This is allowed only if the already attached *ResourceRefCount* can be closed. The reference count of the *ResourceRefCount* being attached will be incremented.

Parameters

hdulist_refcnt : ResourceRefCount, None

A *ResourceRefCount* object containing a `astropy.io.fits.HDUList` object. If it is `None`,it will release and close the attached *ResourceRefCount* object.

Raises

ValueError

The (already) attached *ResourceRefCount* must have reference count <= 1 so that it can be closed before being replaced with a new resource.

ValueError

The resource being attached must be in an open state.

closed
Is the *ImageRef* object closed?

extname
Extension name of the first extension.

hdu
`astropy.io.fits.HDUList` of the attached image.

refcount
Reference count of the attached `astropy.io.fits.HDUList`.

stsci.skypac.utils.**is_countrate**(*hdulist*, *ext*, *units_kwd='BUNIT'*, *guess_if_missing=True*, *telescope=None*, *instrument=None*, *verbose=True*, *flog=None*)

Infer the units of the data of the input image from the input image. Specifically, it tries to infer whether the units are counts (or count-like) or if the units are count-`rate`.

The units of data are determined from the `BUNIT` header keyword by searching its value for the division sign '/'. If the division sign is not found, then the units are assumed to be "counts". If the division sign is found in the `BUNIT` value and if the numerator is one of the following: 'ELECTRONS','COUNTS', or 'DN', and denumerator is either 'S','SEC', or 'SECOND', then the units are assumed to be count-rate.

Parameters

hdulist : `astropy.io.fits.HDUList`

`astropy.io.fits.HDUList` of the image.

ext : tuple, int, str

Extension specification for whose data the units need to be inferred. An int `ext` specifies extension number. A tuple in the form (str, int) specifies extension name and number. A string `ext` specifies extension name and the extension version is assumed to be 1. See documentation for `astropy.io.fits.getData` for examples.

units_kwd : str (Default = 'BUNIT')

FITS header keyword describing data units of the image. This keyword is assumed to be in the header of the extension specified by the `ext` parameter.

guess_if_missing : bool (Default = True)

Instructs to try make best guess on image units when the keyword specified by `units_kwd` is not found in the image header. The first action will be to look for this keyword in the primary header, and if not found, infer the units based on the telescope, instrument, and detector information.

telescope : str, None (Default = None)

Specifies the telescope from which the data came. If not specified, the value specified in the `TELESCOP` keyword in the primary header will be used.

instrument : str, None (Default = None)

Specifies the instrument used for acquiring data. If not specified, the value specified in the `INSTRUME` keyword in the primary header will be used.

verbose : bool (Default = True)

Specifies whether to print warning messages.

flog : str, file, MultiFileLog, None (Default = None)

Specifies the log file to which the messages should be printed. It can be a file name, a file object, a MultiFileLog object, or None.

Returns

bool, None

Returns `True` if the units of the input image for a given extension are count-rate-like and `False` if the units are count-like. Returns `None` if the units cannot be inferred from the header.

`stsci.skypac.utils.`**`ext2str`**(*ext*, *compact=False*, *default_extver=1*)

Return a string representation of an extension specification.

Parameters

ext : tuple, int, str

Extension specification can be a tuple of the form (str,int), e.g., ('sci',1), an integer (extension number), or a string (extension name).

compact : bool (Default = False)

If `compact=True` the returned string will have extension name quoted and separated by a comma from the extension number, e.g., "'sci',1". If `compact=False` the returned string will have extension version immediately follow the extension name, e.g., 'sci1'.

default_extver : int (Default = 1)

Specifies the extension version to be used when the `ext` parameter is a string (extension name).

Returns

strext : str

String representation of extension specification `ext`.

Raises

TypeError

Unexpected extension type.

Examples

```
>>> ext2str('sci',compact=False,default_extver=6)
"'sci',6"
>>> ext2str(('sci',2))
"'sci',2"
>>> ext2str(4)
'4'
>>> ext2str('dq')
"'dq',1"
>>> ext2str('dq',default_extver=2)
"'dq',2"
>>> ext2str('sci',compact=True,default_extver=2)
'sci2'
```

`stsci.skypac.utils.`**`openImageEx`**(*filename*, *mode='readonly'*, *dqmode='readonly'*, *memmap=True*, *saveAsMEF=True*, *output_base_fitsname=None*, *clobber=True*, *imageOnly=False*, *openImageHDU=True*, *openDQHDU=False*, *preferMEF=True*, *verbose=False*)

Open an image file and (if requested) the associated DQ file and return corresponding *`ImageRef`* objects.

This function is an enhanced version of `stsci.tools.fileutil.openImage()` function in that it can open both the image file and the associated DQ image. It also provides additional inormation about the opened

files: file type, original file name, DQ model ("intrinsic", where DQ data are placed in the same file as the science data, or "extrinsic" when DQ data are in a separate file from the science data), etc. Because of the way it was implemented, it requires half of the number of calls to `astropy.io.fits.open` thus making it almost twice as fast as the `openImage()` function.

Parameters

filename : str

File name of the file to be opened. The image file formats are recognized: simple/MEF FITS, HST GEIS format, or WAIVER FITS format.

mode : str (Default = 'readonly')

File mode used to open main image FITS file. See `astropy.io.fits.open` for more details.

dqmode : str (Default = 'readonly')

File mode used to open DQ image FITS file. See parameter `mode` above for more details.

memmap : bool (Default = True)

Should memory mapping to be used whe opening simple/MEF FITS?

saveAsMEF : bool (Default = True)

Should an input GEIS or WAIVER FITS be converted to simple/MEF FITS?

output_base_fitsname : str, None (Default = None)

The base name of the output simple/MEF FITS when `saveAsMEF` is `True`. If it is `None`, the file name of the converted file will be determined according to HST file naming conventions.

clobber : bool (Default = True)

If `saveAsMEF` is `True`, should any existing output files be overwritten?

imageOnly : bool (Default = False)

Should this function open the image file only? If `True`, then the DQ-related attributes will not be valid.

openImageHDU : bool (Default = True)

Indicates whether the returned *`ImageRef`* object corresponding to the science image file should be in an open or closed state.

openDQHDU : bool (Default = False)

Indicates whether the returned *`ImageRef`* object corresponding to the DQ image file should be in an open or closed state.

preferMEF : bool (Default = True)

Should this function open an existing MEF file that complies with HST naming convention when the input file is in GEIS or WAIVER FITS format, even when `saveAsMEF=False` or `clobber=False`?

verbose : bool (Default = True)

If `True`, some addition information will be printed out to standard output.

Returns

(img, dqimg) : (ImageRef, ImageRef)

A tuple of *ImageRef* objects corresponding to the science image and to the DQ image. Each object in the returned tuple open or closed depending on the input arguments.

Note: If the returned object is *open*, then its reference count will be at least 1. The caller is responsible for "releasing" the object when it is no longer needed.

Note: If the DQ model of the opened file is "intrinsic", then the `dqimg` component of the returned tuple will hold a reference counter to the same image. Thus, for "intrinsic" DQ data models, the reference count of the returned objects may be 2 (if both science `openImageHDU` and `openDQHDU` are `True`).

Raises

ValueError

If input file is neither a GEIS file nor a FITS file.

ValueError

Errors occured while accessing/reading the file possibly due to corrupted file, non-compliant file format, etc.s

`stsci.skypac.utils.`**`count_extensions`**(*img*, *extname='SCI'*)

Return the number of `extname` extensions in the input image `img`. If `extname` is `None`, return the number of all image-like extensions.

Input parameters are identical to those of *get_extver_list()*.

See also:

get_extver_list, *get_ext_list*

Examples

```
>>> count_extensions('j9irw1rqq_flt.fits',extname='SCI')
2
>>> count_extensions('j9irw1rqq_flt.fits',extname=None)
10
```

`stsci.skypac.utils.`**`get_ext_list`**(*img*, *extname='SCI'*)

Return a list of all extension versions of `extname` extensions. `img` can be either a file name or a `astropy.io.fits.HDUList` object.

This function is similar to *get_extver_list()*, the main difference being that it returns a list of fully qualified extensions: either tuples of the form (`extname`,`extver`) or integer extension numbers (when `extname=None`).

See also:

get_extver_list, *count_extensions*

Examples

```
>>> get_ext_list('j9irw1rqq_flt.fits',extname='SCI')
[('SCI', 1), ('SCI', 2)]
>>> get_ext_list('j9irw1rqq_flt.fits',extname=None)
[1, 2, 3, 4, 5, 6, 8, 9, 10, 11]
```

stsci.skypac.utils.**get_extver_list**(*img*, *extname='SCI'*)

Return a list of all extension versions with `extname` extension names. If `extname` is `None`, return extension **numbers** of all image-like extensions.

Note: If input image is a `ImageRef`, this function will **not** modify its reference count.

Parameters

img : str, `astropy.io.fits.HDUList`, or `ImageRef`

Input image object. If `img` is a string object (file name) then that file will be opened. If the file pointed to by the file name is a GEIS or WAIVER FITS file, it will be converted to a simple/MEF FITS format if `clobber '='True`.

extname : str (Default = 'SCI')

Indicates extension *name* for which all existing extension *versions* should be found. If `extname '='None`, then `get_extver_list` will return a list of extension *numbers* of all image-like extensions.

Returns

extver : list

List of extension versions corresponding to the input `extname`. If `extname '='None`, it will return a list of extension *numbers* of all image-like extensions.

Raises

IOError

Unable to open input image file.

TypeError

Argument `img` must be either a file name (str), an *ImageRef*, or a `astropy.io.fits.HDUList` object.

TypeError

Argument `extname` must be either a string or None.

See also:

get_ext_list, *count_extensions*

Examples

```
>>> get_extver_list('j9irw1rqq_flt.fits',extname='sci')
[1, 2]
>>> get_extver_list('j9irw1rqq_flt.fits',extname=None)
[1, 2, 3, 4, 5, 6, 8, 9, 10, 11
```

stsci.skypac.utils.**file_name_components**(*fname*, *detect_HST_FITS_suffix=True*)

Splits base file name into a root, suffix, and extension. Given a full path, this function extracts the base name, and splits it into three components: root name, suffix, and file extension.

Parameters

fname : str

file name

detect_HST_FITS_suffix : bool (Default = True)

If True, detects the suffix of most HST files by looking for the rightmost occurence of the underscore ('_') in the file name.

Returns

root : str

Root name of the file. When `detect_HST_FITS_suffix=True`, this is the part of the file name *preceding* the rightmost suffix separator ('_'). Otherwise, it is the base file name without file extension.

suffix : str

If `detect_HST_FITS_suffix=True`, this field will contain the suffix of most HST files, i.e., the part of the file name contained between the rightmost suffix separator ('_') and file extension separator. This return value will be an empty string if If `detect_HST_FITS_suffix=False` or if the file name has no extension separator.

fext : str

File extension

Examples

```
>>> file_name_components('/data/m87/ua0x5001m_c0f.fits')
('ua0x5001m', 'c0f', '.fits')
>>> file_name_components('/data/m87/ua0x5001m_c0f.fits',False)
('ua0x5001m_c0f', '', '.fits')
```

stsci.skypac.utils.**temp_mask_file**(*rootname*, *suffix*, *ext*, *data*, *dir=os.path.curdir*, *fnameOnly=False*)

Saves 2D data array to temporary simple FITS file. The name of the emporary file is generated based on the input parameters.

Parameters

data : numpy array

Data to be written to the temporary FITS file. Data will be written in the primary HDU.

rootname : str

Root name of the file.

prefix : str (Default = 'tmp')

Prefix to be added in front of the root name. If `randomize_prefix` is `True`, then a random string will be added to the right of the string specified by `prefix` (with no separator between them). Prefix (or the randomized prefix) will be separated from the root name by the string specified in `sep`. If prefix is an empty string (`''`) then no prefix will be prepended to the root file name.

suffix : str (Default = 'mask')

Suffix to be added to the root name. Suffix will be separated from the root name by the string specified in `sep`.

ext : int, str, or tuple of the form (str, int)

Extention to be added to the temporary file *after* the suffix. Extension name string will be separated from the suffix by the string specified in `sep`.

sep : str (Default = '_')

Separator string to be inserted between (randomized) prefix and root name, root name and suffix, and suffix and extension.

randomize_prefix : bool (Default = True)

Specifies whether to add (postpend) a random string to string specified by `prefix`.

dir : str (Default = os.path.curdir)

Directory to which the temporary file should be written. If directory `dir=None` then the file will be written to the default (for more details, see the explanation for argument `dir` to the tempfile.mstemp function).

fnameOnly : bool (Default = False)

Specifies what should *`temp_mask_file`* return: file name of the created file (if `fnameOnly=True`), or a tuple with the file name of the created file and an open `ImageRef` object of that file.

Returns

fname : str

File name of the temporary file.

mask : ImageRef

An open `ImageRef` object of the temporary FITS file. This is returned as a tuple together with the file name only when `fnameOnly=False`.

Note: Mask data will be in the Primary HDU.

Raises

TypeError

Extension specifier must be either an integer, a string, or a tuple of the form (str, int).

Examples

```
>>> import numpy np
>>> from stsci import skypac
>>> mask=np.ones((800,800),dtype=np.uint8)
>>> skypac.utils.temp_mask_file(mask, 'ua0x5001m',
...     suffix='skymatch_mask', ext=('sci',4), dir='/data/m87',
...     fnameOnly=True)
'/data/m87/tmp39gCpw_ua0x5001m_skymatch_mask_sci4.fits'
>>> skypac.utils.temp_mask_file(mask, 'ua0x5001m',
...     suffix='skymatch_mask', ext=('sci',4), dir='.', fnameOnly=True)
'tmpxl7LTO_ua0x5001m_skymatch_mask_sci4.fits'
>>> skypac.utils.temp_mask_file(mask, 'ua0x5001m',
...     suffix='skymatch_mask', ext=('sci',4), dir='.',fnameOnly=False)
('tmpxMcL5g_ua0x5001m_skymatch_mask_sci4.fits', <skypac.utils.ImageRef object at 0x101f5a3d0>)
```

`stsci.skypac.utils.`**`almost_equal`**(*arr1*, *arr2*, *fp_accuracy=None*, *fp_precision=None*)

Compares two values, or values of `ndarray` and verifies that these values are close to each other. For exact type (integers and boolean) the comparison is exact. For inexact types (`float`, `numpy.float32`, etc.) it checks that the values (or *all* values in a `numpy.ndarray`) satisfy the following inequality:

$$|v1 - v2| \leq a + 10^{-p} \cdot |v2|,$$

where `a` is the accuracy ("absolute error") and `p` is precision ("relative error").

Parameters

arr1 : float, int, bool, str, numpy.ndarray, None, etc.

First value or array of values to be compared.

arr2 : float, int, bool, str, numpy.ndarray, None, etc.

Second value or array of values to be compared.

fp_accuracy : int, float, None (Default = None)

Accuracy to withing values should be close. Default value will use twice the value of the machine accuracy (machine epsilon) for the input type. This parameter has effect only when the values to be compared are of inexact type (e.g., `float`).

fp_precision : int, float, None (Default = None)

Accuracy to withing values should be close. Default value will use twice the value of the machine precision (resolution) for the input type. This parameter has effect only when the values to be compared are of inexact type (e.g., `float`).

Returns

almost_equal : bool

Returns `True` if input values are close enough to each other or `False` otherwise.

Raises

ValueError

If `fp_accuracy` is negative.

TypeError

If `fp_precision` is not `int`, `float`, or `None`.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

S

M

O

P

R

S

T

U

W