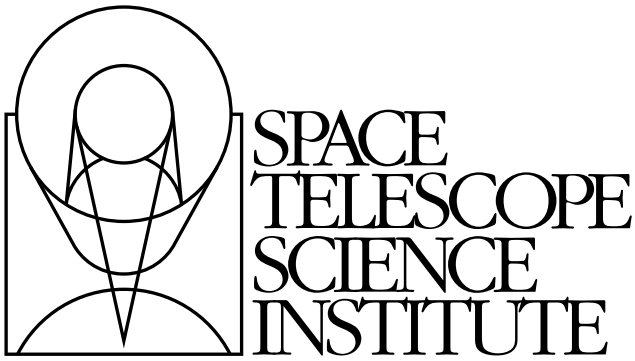


# acstools Documentation

***Release 2.0.0***

**STScI**

April 18, 2016

CONTENTS

Modules for Advanced Camera for Surveys (ACS).

http://www.stsci.edu/hst/acs/

**Note:** Standalone CTE correction (`PixCteCorr`) is no longer supported. Please use *ACSCTE*.

Contents:

CHAPTER

ONE

# CALACS

The calacs module contains a function *calacs* that calls the CALACS executable. Use this function to facilitate batch runs of CALACS, or for the TEAL interface.

## 1.1 Examples

In Python without TEAL:

```
>>> from acstools import calacs
>>> calacs.calacs(filename)
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import calacs
>>> teal.teal('calacs')
```

In Pyraf:

```
--> import acstools
--> epar calacs
```

acstools.calacs.**calacs**(*input_file*, *exec_path=None*, *time_stamps=False*, *temp_files=False*, *verbose=False*, *debug=False*, *quiet=False*, *single_core=False*)

Run the calacs.e executable as from the shell.

By default this will run the calacs given by 'calacs.e'.

**Parameters**

**input_file** : str

Name of input file.

**exec_path** : str, optional

The complete path to a calacs executable.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**temp_files** : bool, optional

Set to True to have CALACS save temporary files.

**verbose** : bool, optional

Set to True for verbose output.

**debug** : bool, optional

Set to True to turn on debugging output.

**quiet** : bool, optional

Set to True for quiet output.

**single_core** : bool, optional

CTE correction in CALACS will by default try to use all available CPUs on your computer. Set this to True to force the use of just one CPU.

CHAPTER

TWO

# CALACS.E (HSTCAL)

A detailed description of this new and improved CALACS is available in ACS Data Handbook v7.0 or later. If you have questions not answered in the documentation, please contact STScI Help Desk (`help[at]stsci.edu`).

## 2.1 Running CALACS

### 2.1.1 Where to Find CALACS

CALACS is now part of HSTCAL package, which can be downloaded from STSDAS download page.

### 2.1.2 Usage

From the command line:

```
calacs.e jb1f89eaq_raw.fits [command line options]
```

### 2.1.3 Command Line Options

CALACS supports several command line options:

- -t
  - Print verbose time stamps.
- -s
  - Save temporary files.
- -v
  - Turn on verbose output.
- -d
  - Turn on debug output.
- -q
  - Turn on quiet output.
- -1
  - Turn off parallel processing for PCTECORR. Try this option if you encounter problems running CALACS with PCTECORR=PERFORM.

### 2.1.4 Parallel Processing with OpenMP

By default, CALACS will attempt to perform PCTECORR using all available CPUs on your machine. You can set the maximum number of CPUs available for CALACS by setting the `OMP_NUM_THREADS` environmental variable.

In tcsh:

```
setenv OMP_NUM_THREADS 2
```

In bash:

```
export OMP_NUM_THREADS=2
```

### 2.1.5 Batch CALACS

The recommended method for running CALACS in batch mode is to use Python and the `acstools` package in STSDAS distribution.

For example:

```
from acstools import calacs
import glob

for fits in glob.iglob('j*_raw.fits'):
    calacs.calacs(fits)
```

## 2.2 BIASCORR

BIASCORR is now performed before BLEVCORR. This should not significantly affect science results. This change was necessary to accomodate BIASFILE subtraction in DN with the rest of the calculations done in ELECTRONS.

## 2.3 Unit Conversion to Electrons

The image is multiplied by gain right after BIASCORR, converting it to ELECTRONS. This step is no longer embedded within FLATCORR.

## 2.4 BLEVCORR

BLEVCORR is now performed after BIASCORR. Calculations are done in ELECTRONS.

For post-SM4 full-frame WFC exposures, it also includes:

- de-striping to remove stripes introduced by new hardware installed during SM-4 (J. Anderson; ACS ISR 2011-05); and
- if JWROTYPE=DS_int and CCDGAIN=2, also correct for bias shift (ACS ISR 2012-02) and cross-talk (N. Grogin; ACS ISR 2010-02).

## 2.5 Pixel-Based CTE Correction (PCTECORR)

For all full-frame WFC exposures, pixel-based CTE correction (ACS ISR 2010-03 and 2012-03) is applied in ACSCTE that occurs after the ACSCCD series; i.e., after BLEVCORR.

Because the CTE correction is applied before DARKCORR and FLSHCORR, it is necessary to use a CTE-corrected dark (DRKCFILE) if the PCTECORR step is enabled.

Parameters characterizing the CTE correction are stored in a reference table, PCTETAB.

---

**Note:** CALACS 8.2 and later uses a slightly different PCTETAB format, where the COL_SCALE extension does not include overscan columns.

---

### 2.5.1 Required Keywords

Running CALACS with pixel-based CTE correction requires the following header keywords:

- PCTECORR
  - By default, set to PERFORM for all full-frame WFC exposures, except BIAS.
- PCTETAB
  - Reference table containing CTE correction parameters. By default, it should be in the `jref` directory and have the suffix `_cte.fits`.
- DRKCFILE
  - Similar to DARKFILE but with CTE correction performed. By default, it should be in the `jref` directory and have the suffix `_dkc.fits`. This is necessary because PCTECORR is done before DARKCORR.

### 2.5.2 Optional Keywords

You may adjust some CTE correction algorithm parameters by changing the following keywords in RAW image header. The default values are picked for optimum results in a typical WFC full-frame exposure. Changing these values is not recommended unless you know what you are doing.

- PCTENSMD
  - Read noise mitigation mode:
    - 0 - No mitigation
    - 1 - Perform noise smoothing
    - 2 - No noise smoothing
  - Overwrites NSEMODEL in PCTETAB.
- PCTERNCL
  - Read noise level of image in ELECTRONS. This is not used if you specified no mitigation in read noise mitigation mode.
  - Overwrites RN_CLIP in PCTETAB.
- PCTETRSH
  - Over-subtraction correction threshold. Pixel below this value in ELECTRONS after CTE correction is considered over-corrected and will re-corrected with smaller correction.

  - Overwrites SUBTHRSH in PCTETAB.
- PCTESMIT
  - Number of iterations of readout simulation per column.
  - Overwrites SIM_NIT in PCTETAB.
- PCTESHFT
  - Number of shifts each readout simulation is broken up into.
  - Overwrites SHFT_NIT in PCTETAB.

## 2.6 Dark Current Subtraction (DARKCORR)

It uses DARKFILE if PCTECORR=OMIT, otherwise it uses DRKCFILE (CTE-corrected dark reference file).

Dark image is now scaled by EXPTIME and FLASHDUR. For post-SM4 non-BIAS WFC images, extra 3 seconds are also added to account for idle time before readout. Any image with non-zero EXPTIME is considered not a BIAS.

## 2.7 Post-Flash Correction (FLSHCORR)

Post-flash correction is now performed after DARKCORR in the ACS2D step. When FLSHCORR=PERFORM, it uses FLSHFILE.

## 2.8 FLATCORR

Conversion from DN to ELECTRONS no longer depends on FLATCORR=PERFORM. Unit conversion is done for all exposures after BIASCORR.

## 2.9 Photometry Keywords (PHOTCORR)

The PHOTCORR step is now performed using tables of precomputed values instead of calls to SYNPHOT. The correct table for a given image must be specified in the IMPHTTAB header keyword in order for CALACS to perform the PHOTCORR step. By default, it should be in the `jref` directory and have the suffix `_imp.fits`. Each DETECTOR uses a different table.

If you do not wish to use this feature, set PHOTCORR to OMIT.

## 2.10 CALACS Output

Using RAW as input:

- flt.fits: Same as existing FLT.
- flc.fits: Similar to FLT, except with pixel-based CTE correction applied.

Using ASN as input with ACSREJ:

- crj.fits: Same as existing CRJ.

- crc.fits: Similar to CRJ, except with pixel-based CTE correction applied.

CALACS uses HSTIO that utilizes `PIXVALUE` keyword to represent a data extension with constant value. However, this is not a standard FITS behavior and is not recognized by PyFITS. Therefore, one should use `stsci.tools.stpyfits`, which is distributed as part of `stsci_python`, instead of `pyfits` or `astropy.io.fits` when working with CALACS products. To use `stpyfits` in Python:

```
from stsci.tools import stpyfits as pyfits
```

CHAPTER

THREE

# ACSCCD

The acsccd module contains a function *acsccd* that calls the ACSCCD executable. Use this function to facilitate batch runs of ACSCCD, or for the TEAL interface.

**Note:** Calibration flags are controlled by primary header.

**Warning:** Do not use with SBC MAMA images.

## 3.1 Examples

In Python without TEAL:

```
>>> from acstools import acsccd
>>> acsccd.acsccd('*raw.fits')
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import acsccd
>>> teal.teal('acsccd')
```

In Pyraf:

```
--> import acstools
--> epar acsccd
```

acstools.acsccd.**acsccd**(*input*, *exec_path=''*, *time_stamps=False*, *verbose=False*, *quiet=False*)

Run the acsccd.e executable as from the shell.

Expect input to be `*_raw.fits`. Output is automatically named `*_blv_tmp.fits`.

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a single filename ('j1234567q_raw.fits')
- a Python list of filenames
- a partial filename with wildcards ('*raw.fits')
- filename of an ASN table ('j12345670_asn.fits')

- an at-file (`@input`)

**exec_path** : str, optional

The complete path to ACSCCD executable. If not given, run ACSCCD given by 'acsccd.e'.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**verbose** : bool, optional

Set to True for verbose output.

**quiet** : bool, optional

Set to True for quiet output.

CHAPTER

FOUR

# ACSCTE

The acscte module contains a function *acscte* that calls the ACSCTE executable. Use this function to facilitate batch runs of ACSCTE, or for the TEAL interface.

Only WFC full-frame and some 2K subarrays are currently supported. See ACS Data Handbook for more details.

**Note:** Calibration flags are controlled by primary header.

## 4.1 Examples

In Python without TEAL:

```
>>> from acstools import acscte
>>> acscte.acscte('*blv_tmp.fits')
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import acscte
>>> teal.teal('acscte')
```

In Pyraf:

```
--> import acstools
--> epar acscte
```

acstools.acscte.**acscte**(*input*, *exec_path=''*, *time_stamps=False*, *verbose=False*, *quiet=False*, *single_core=False*)

Run the acscte.e executable as from the shell.

Expect input to be `*_blv_tmp.fits`. Output is automatically named `*_blc_tmp.fits`.

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a single filename ('j1234567q_blv_tmp.fits')
- a Python list of filenames
- a partial filename with wildcards ('*blv_tmp.fits')
- filename of an ASN table ('j12345670_asn.fits')
- an at-file (`@input`)

**exec_path** : str, optional

The complete path to ACSCTE executable. If not given, run ACSCTE given by 'acscte.e'.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**verbose** : bool, optional

Set to True for verbose output.

**quiet** : bool, optional

Set to True for quiet output.

**single_core** : bool, optional

CTE correction in ACSCTE will by default try to use all available CPUs on your computer. Set this to True to force the use of just one CPU.

CHAPTER

# FIVE

# ACSREJ

The acsrej module contains a function *acsrej* that calls the ACSREJ executable. Use this function to facilitate batch runs of ACSREJ, or for the TEAL interface.

## 5.1 Examples

In Python without TEAL:

```
>>> from acstools import acsrej
>>> acsrej.acsrej('*flt.fits', 'combined_image.fits')
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import acsrej
>>> teal.teal('acsrej')
```

In Pyraf:

```
--> import acstools
--> epar acsrej
```

acstools.acsrej.**acsrej**(*input*, *output*, *exec_path=''*, *time_stamps=False*, *verbose=False*, *shadcorr=False*, *crrejtab=''*, *crmask=False*, *scalense=None*, *initgues=''*, *skysub=''*, *crsigmas=''*, *crradius=None*, *crthresh=None*, *badinpdq=None*, *newbias=False*)

Run the acsrej.e executable as from the shell.

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a Python list of filenames
- a partial filename with wildcards ('*flt.fits')
- filename of an ASN table ('j12345670_asn.fits')
- an at-file (`@input`)

**output** : str

Output filename.

**exec_path** : str, optional

The complete path to ACSREJ executable. If not given, run ACSREJ given by 'acsrej.e'.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**verbose** : bool, optional

Set to True for verbose output.

**shadcorr** : bool, optional

Perform shutter shading correction. If this is False but SHADCORR is set to PERFORM in the header of the first image, the correction will be applied anyway. Only use this with CCD image, not SBC MAMA.

**crrejtab** : str, optional

CRREJTAB to use. If not given, will use CRREJTAB given in the primary header of the first input image.

**crmask** : bool, optional

Flag CR-rejected pixels in input files. If False, will use CRMASK value in CRREJTAB.

**scalense** : float, optional

Multiplicative scale factor (in percents) applied to noise. Acceptable values are 0 to 100, inclusive. If None, will use SCALENSE from CRREJTAB.

**initgues** : {'med', 'min'}, optional

Scheme for computing initial-guess image. If not given, will use INITGUES from CRREJTAB.

**skysub** : {'none', 'mode'}, optional

Scheme for computing sky levels to be subtracted. If not given, will use SKYSUB from CRREJTAB.

**crsigmas** : str, optional

Cosmic ray rejection thresholds given in the format of 'sig1,sig2,...'. Number of sigmas given will be the number of rejection iterations done. At least 1 and at most 20 sigmas accepted. If not given, will use CRSIGMAS from CRREJTAB.

**crradius** : float, optional

Radius (in pixels) to propagate the cosmic ray. If None, will use CRRADIUS from CRREJTAB.

**crthresh** : float, optional

Cosmic ray rejection propagation threshold. If None, will use CRTHRESH from CRREJTAB.

**badinpdq** : int, optional

Data quality flag used for cosmic ray rejection. If None, will use BADINPDQ from CRREJTAB.

**newbias** : bool, optional

ERR is just read noise, not Poisson noise. This is used for BIAS images.

CHAPTER

SIX

# ACS2D

The acs2d module contains a function *acs2d* that calls the ACS2D executable. Use this function to facilitate batch runs of ACS2D, or for the TEAL interface.

## 6.1 Examples

In Python without TEAL:

```
>>> from acstools import acs2d
>>> acs2d.acs2d('*blv_tmp.fits')
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import acs2d
>>> teal.teal('acs2d')
```

In Pyraf:

```
--> import acstools
--> epar acs2d
```

acstools.acs2d.**acs2d**(*input*, *exec_path=''*, *time_stamps=False*, *verbose=False*, *quiet=False*)

Run the acs2d.e executable as from the shell.

Output is automatically named based on input suffix:

| INPUT | OUTPUT | EXPECTED DATA |
|---|---|---|
| *_raw.fits | *_flt.fits | SBC image. |
| *_blv_tmp.fits | *_flt.fits | ACSCCD output. |
| *_blc_tmp.fits | *_flc.fits | ACSCCD output with PCTECORR. |
| *_crj_tmp.fits | *_crj.fits | ACSREJ output. |
| *_crc_tmp.fits | *_crc.fits | ACSREJ output with PCTECORR. |

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a single filename ('j1234567q_blv_tmp.fits')
- a Python list of filenames
- a partial filename with wildcards ('*blv_tmp.fits')
- filename of an ASN table ('j12345670_asn.fits')

- an at-file (`@input`)

**exec_path** : str, optional

The complete path to ACS2D executable. If not given, run ACS2D given by 'acs2d.e'.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**verbose** : bool, optional

Set to True for verbose output.

**quiet** : bool, optional

Set to True for quiet output.

CHAPTER

SEVEN

# ACSSUM

The acssum module contains a function *acssum* that calls the ACSSUM executable. Use this function to facilitate batch runs of ACSSUM, or for the TEAL interface.

## 7.1 Examples

In Python without TEAL:

```
>>> from acstools import acssum
>>> acssum.acssum('*flt.fits', 'combined_image.fits')
```

In Python with TEAL:

```
>>> from stsci.tools import teal
>>> from acstools import acssum
>>> teal.teal('acssum')
```

In Pyraf:

```
--> import acstools
--> epar acssum
```

acstools.acssum.**acssum**(*input*, *output*, *exec_path=''*, *time_stamps=False*, *verbose=False*, *quiet=False*)

Run the acssum.e executable as from the shell.

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a Python list of filenames
- a partial filename with wildcards ('*flt.fits')
- filename of an ASN table ('j12345670_asn.fits')
- an at-file (`@input`)

**output** : str

Output filename. If *output* is '' and *input* is '*_asn.fits', *output* will be automatically set to '*_sfl.fits'. Otherwise, it is an error not to provide a specific *output*.

**exec_path** : str, optional

The complete path to ACSSUM executable. If not given, run ACSSUM given by 'acssum.e'.

**time_stamps** : bool, optional

Set to True to turn on the printing of time stamps.

**verbose** : bool, optional

Set to True for verbose output.

**quiet** : bool, optional

Set to True for quiet output.

CHAPTER

EIGHT

# ACS DESTRIPE

Remove horizontal stripes from ACS WFC post-SM4 data.

For more information, see Removal of Bias Striping Noise from Post-SM4 ACS WFC Images.

**Note:**

- Does not work on RAW image.
- Uses the flatfield specified by the image header keyword PFLTFILE. If keyword value is 'N/A', as is the case with biases and darks, then unity flatfield is used.
- Uses post-flash image specified by the image header keyword FLSHFILE. If keyword value is 'N/A', then dummy post-flash with zeroes is used.
- Uses the dark image specified by the image header keyword DARKFILE. If keyword value is 'N/A', then dummy dark with zeroes is used.

## 8.1 Examples

In Python without TEAL:

```
>>> from acstools import acs_destripe
>>> acs_destripe.clean('uncorrected_flt.fits', 'csck',
...                    mask1='mymask_sci1.fits', mask2='mymask_sci2.fits',
...                    clobber=False, maxiter=15, sigrej=2.0)
```

In Python with TEAL:

```
>>> from acstools import acs_destripe
>>> from stsci.tools import teal
>>> teal.teal('acs_destripe')
```

In Pyraf:

```
--> import acstools
--> teal acs_destripe
```

From command line:

```
% acs_destripe [-h] [--stat STAT] [--lower [LOWER]] [--upper [UPPER]]
               [--binwidth BINWIDTH] [--mask1 MASK1] [--mask2 MASK2]
               [--dqbits [DQBITS]] [--rpt_clean RPT_CLEAN]
```

```
              [--atol [ATOL]] [-c] [-q] [--version]
              input suffix [maxiter] [sigrej]
```

acstools.acs_destripe.**clean**(*input*, *suffix*, *stat='pmode1'*, *maxiter=15*, *sigrej=2.0*, *lower=None*, *upper=None*, *binwidth=0.3*, *mask1=None*, *mask2=None*, *dqbits=None*, *rpt_clean=0*, *atol=0.01*, *clobber=False*, *verbose=True*)

Remove horizontal stripes from ACS WFC post-SM4 data.

**Parameters**

**input** : str or list of str

Input filenames in one of these formats:

- a Python list of filenames
- a partial filename with wildcards ('*flt.fits')
- filename of an ASN table ('j12345670_asn.fits')
- an at-file (`@input`)

**suffix** : str

The string to use to add to each input file name to indicate an output product. This string will be appended to the suffix in each input filename to create the new output filename. For example, setting *suffix='csck'* will create '*_csck.fits' images.

**stat** : { 'pmode1', 'pmode2', 'mean', 'mode', 'median', 'midpt' } (Default = 'pmode1')

Specifies the statistics to be used for computation of the background in image rows:

- 'pmode1' - SEXTRACTOR-like mode estimate based on a modified Pearson's rule: `2.5*median-1.5*mean`;
- 'pmode2' - mode estimate based on Pearson's rule: `3*median-2*mean`;
- 'mean' - the mean of the distribution of the "good" pixels (after clipping, masking, etc.);
- 'mode' - the mode of the distribution of the "good" pixels;
- 'median' - the median of the distribution of the "good" pixels;
- 'midpt' - estimate of the median of the distribution of the "good" pixels based on an algorithm similar to IRAF's *imagestats* task (`CDF(midpt)=1/2`).

---

**Note:** The midpoint and mode are computed in two passes through the image. In the first pass the standard deviation of the pixels is calculated and used with the *binwidth* parameter to compute the resolution of the data histogram. The midpoint is estimated by integrating the histogram and computing by interpolation the data value at which exactly half the pixels are below that data value and half are above it. The mode is computed by locating the maximum of the data histogram and fitting the peak by parabolic interpolation.

---

**maxiter** : int

This parameter controls the maximum number of iterations to perform when computing the statistics used to compute the row-by-row corrections.

**sigrej** : float

This parameters sets the sigma level for the rejection applied during each iteration of statistics computations for the row-by-row corrections.

**lower** : float, None (Default = None)

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

**binwidth** : float (Default = 0.1)

Histogram's bin width, in sigma units, used to sample the distribution of pixel brightness values in order to compute the background statistics. This parameter is aplicable *only* to *stat* parameter values of *'mode'* or *'midpt'*.

**clobber** : bool

Specify whether or not to 'clobber' (delete then replace) previously generated products with the same names.

**mask1** : str, numpy.ndarray, None, or list of these types

Mask images for `SCI,1`, one for each input file. Pixels with zero values will be masked out, in addition to clipping.

**mask2** : str, numpy.ndarray, None, or list of these types

Mask images for `SCI,2`, one for each input file. Pixels with zero values will be masked out, in addition to clipping. This is not used for subarrays.

**dqbits** : int, str, None (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 de-striping 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 de-striping computations, 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.

Set *dqbits* to 0 to make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels not to be used for de-striping computations.

Default value (*None*) will turn off the use of image's DQ array for de-striping computations.

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 sky computations 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.

---

**Note:** DQ masks (if used), *will be* combined with user masks specified in the *mask1* and *mask2* parameters (if any).

---

**rpt_clean** : int

An integer indicating how many *additional* times stripe cleaning should be performed on the input image. Default = 0.

**atol** : float, None

The threshold for maximum absolute value of bias stripe correction below which repeated cleanings can stop. When *atol* is *None* cleaning will be repeated *rpt_clean* number of times. Default = 0.01 [e].

**verbose** : bool

Print informational messages. Default = True.

## 8.2 Global Variables

acstools.acs_destripe.**MJD_SM4 = 54967**
int(x=0) -> int or long int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

## 8.3 Version

acstools.acs_destripe.**__version__ = '0.8.0'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

acstools.acs_destripe.**__vdate__ = '31-Mar-2015'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

## 8.4 Author

acstools.acs_destripe.**__author__ = 'Norman Grogin, STScI, March 2012.'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

CHAPTER

NINE

# ACS DESTRIPE PLUS

Fully calibrate post-SM4 ACS/WFC exposures using the standalone *ACS Destripe* tool to remove stripes between ACSCCD and ACSCTE steps in CALACS.

This script runs CALACS (8.3.1 or higher only) and `acs_destripe` on ACS/WFC images. Input files must be RAW full-frame or subarray ACS/WFC exposures taken after SM4. Resultant outputs are science-ready FLT and FLC (if applicable) files.

This script is useful for when built-in CALACS destriping algorithm using overscans is insufficient or unavailable.

For more information, see Removal of Bias Striping Noise from Post-SM4 ACS WFC Images.

## 9.1 Examples

In Python without TEAL:

```
>>> from acstools import acs_destripe_plus
>>> acs_destripe_plus.destripe_plus(
...     'j12345678_raw.fits', suffix='strp', maxiter=15, sigrej=2.0,
...     scimask1='mymask_sci1.fits', scimask2='mymask_sci2.fits',
...     clobber=False, cte_correct=True)
```

In Python with TEAL:

```
>>> from acstools import acs_destripe_plus
>>> from stsci.tools import teal
>>> teal.teal('acs_destripe_plus')
```

In Pyraf:

```
--> import acstools
--> teal acs_destripe_plus
```

From command line:

```
% acs_destripe_plus [-h] [--suffix SUFFIX] [--stat STAT]
                    [--maxiter MAXITER] [--sigrej SIGREJ]
                    [--lower [LOWER]] [--upper [UPPER]]
                    [--binwidth BINWIDTH] [--sci1_mask SCI1_MASK]
                    [--sci2_mask SCI2_MASK] [--dqbits [DQBITS]]
                    [--rpt_clean RPT_CLEAN] [--atol [ATOL]] [--nocte]
                    [--clobber] [-q] [--version]
                    input
```

acstools.acs_destripe_plus.**destripe_plus**(*inputfile*, *suffix='strp'*, *stat='pmode1'*, *maxiter=15*, *sigrej=2.0*, *lower=None*, *upper=None*, *binwidth=0.3*, *scimask1=None*, *scimask2=None*, *dqbits=None*, *rpt_clean=0*, *atol=0.01*, *cte_correct=True*, *clobber=False*, *verbose=True*)

Calibrate post-SM4 ACS/WFC exposure(s) and use standalone *ACS Destripe*.

This takes a RAW image and generates a FLT file containing its calibrated and destriped counterpart. If CTE correction is performed, FLC will also be present.

**Parameters**

**inputfile** : str or list of str

Input filenames in one of these formats:

- a Python list of filenames
- a partial filename with wildcards ('*raw.fits')
- filename of an ASN table ('j12345670_asn.fits')
- an at-file (`@input`)

**suffix** : str

The string to use to add to each input file name to indicate an output product of `acs_destripe`. This only affects the intermediate output file that will be automatically renamed to `*blv_tmp.fits` during the processing.

**stat** : { 'pmode1', 'pmode2', 'mean', 'mode', 'median', 'midpt' } (Default = 'pmode1')

Specifies the statistics to be used for computation of the background in image rows:

- 'pmode1' - SEXTRACTOR-like mode estimate based on a modified Pearson's rule: `2.5*median-1.5*mean`;
- 'pmode2' - mode estimate based on Pearson's rule: `3*median-2*mean`;
- 'mean' - the mean of the distribution of the "good" pixels (after clipping, masking, etc.);
- 'mode' - the mode of the distribution of the "good" pixels;
- 'median' - the median of the distribution of the "good" pixels;
- 'midpt' - estimate of the median of the distribution of the "good" pixels based on an algorithm similar to IRAF's *imagestats* task (`CDF(midpt)=1/2`).

---

**Note:** The midpoint and mode are computed in two passes through the image. In the first pass the standard deviation of the pixels is calculated and used with the *binwidth* parameter to compute the resolution of the data histogram. The midpoint is estimated by integrating the histogram and computing by interpolation the data value at which exactly half the pixels are below that data value and half are above it. The mode is computed by locating the maximum of the data histogram and fitting the peak by parabolic interpolation.

---

**maxiter** : int

This parameter controls the maximum number of iterations to perform when computing the statistics used to compute the row-by-row corrections.

**sigrej** : float

This parameters sets the sigma level for the rejection applied during each iteration of statistics computations for the row-by-row corrections.

**lower** : float, None (Default = None)

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

**binwidth** : float (Default = 0.1)

Histogram's bin width, in sigma units, used to sample the distribution of pixel brightness values in order to compute the background statistics. This parameter is aplicable *only* to *stat* parameter values of *'mode'* or *'midpt'*.

**clobber** : bool

Specify whether or not to 'clobber' (delete then replace) previously generated products with the same names.

**scimask1** : str or list of str

Mask images for *calibrated* `SCI,1`, one for each input file. Pixels with zero values will be masked out, in addition to clipping.

**scimask2** : str or list of str

Mask images for *calibrated* `SCI,2`, one for each input file. Pixels with zero values will be masked out, in addition to clipping. This is not used for subarrays.

**dqbits** : int, str, None (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 de-striping 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 de-striping computations, 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.

Set *dqbits* to 0 to make *all* non-zero pixels in the DQ mask to be considered "bad" pixels, and the corresponding image pixels not to be used for de-striping computations.

Default value (*None*) will turn off the use of image's DQ array for de-striping computations.

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 sky computations 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`.

---

**Note:** DQ masks (if used), *will be* combined with user masks specified in the *scimask1* and *scimask2* parameters (if any).

---

**rpt_clean** : int

An integer indicating how many *additional* times stripe cleaning should be performed on the input image. Default = 0.

**atol** : float, None

The threshold for maximum absolute value of bias stripe correction below which repeated cleanings can stop. When *atol* is *None* cleaning will be repeated *rpt_clean* number of times. Default = 0.01 [e].

**cte_correct** : bool

Perform CTE correction.

**verbose** : bool

Print informational messages. Default = True.

**Raises**

**IOError**

Input file does not exist.

**ValueError**

Invalid header values or CALACS version.

## 9.2 Global Variables

`acstools.acs_destripe_plus.`**SM4_DATE = <Time object: scale='utc' format='iso' value=2008-01-01 00:00:00.000>**

Represent and manipulate times and dates for astronomy.

A `Time` object is initialized with one or more times in the `val` argument. The input times in `val` must conform to the specified `format` and must correspond to the specified time `scale`. The optional `val2` time input should be supplied only for numeric input formats (e.g. JD) where very high precision (better than 64-bit precision) is required.

The allowed values for `format` can be listed with:

```
>>> list(Time.FORMATS)
['jd', 'mjd', 'decimalyear', 'unix', 'cxcsec', 'gps', 'plot_date',
 'datetime', 'iso', 'isot', 'yday', 'fits', 'byear', 'jyear', 'byear_str',
 'jyear_str']
```

**Parameters**

**val** : sequence, str, number, or `Time` object

Value(s) to initialize the time or times.

**val2** : sequence, str, or number; optional

Value(s) to initialize the time or times.

**format** : str, optional

Format of input value(s)

**scale** : str, optional

Time scale of input value(s), must be one of the following: ('tai', 'tcb', 'tcg', 'tdb', 'tt', 'ut1', 'utc')

**precision** : int, optional

Digits of precision in string representation of time

**in_subfmt** : str, optional

Subformat for inputting string times

**out_subfmt** : str, optional

Subformat for outputting string times

**location** : `EarthLocation` or tuple, optional

If given as an tuple, it should be able to initialize an an EarthLocation instance, i.e., either contain 3 items with units of length for geocentric coordinates, or contain a longitude, latitude, and an optional height for geodetic coordinates. Can be a single location, or one for each input time.

**copy** : bool, optional

Make a copy of the input values

acstools.acs_destripe_plus.**SUBARRAY_LIST = ['WFC1-2K', 'WFC1-POL0UV', 'WFC1-POL0V', 'WFC1-POL60V**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

## 9.3 Version

acstools.acs_destripe_plus.**__version__ = '0.4.1'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

acstools.acs_destripe_plus.**__vdate__ = '12-Jan-2016'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

## 9.4 Author

acstools.acs_destripe_plus.**__author__ = 'Leonardo Ubeda, Sara Ogaz (ACS Team), STScI'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

CHAPTER

# TEN

# SATELLITE TRAILS DETECTION

This module contains the tools needed for satellite detection within an ACS/WFC image as published in ACS ISR 2016-01.

**Note:** Only tested for ACS/WFC FLT and FLC images, but it should theoretically work for any instrument.

`skimage.transform.probabilistic_hough_line` gives slightly different results from run to run, but this should not matter since *detsat* only provides crude approximation of the actual trail(s).

Performance is faster when `plot=False`, where applicable.

Currently *not* supported in TEAL and PyRAF.

## 10.1 Examples

```
>>> from acstools.satdet import detsat, make_mask, update_dq
```

Find trail segments for a single image and extension without multiprocessing, and display plots (not shown) and verbose information:

```
>>> results, errors = detsat(
...     'jc8m10syq_flc.fits', chips=[4], n_processes=1, plot=True, verbose=True)
1 file(s) found...
Processing jc8m10syq_flc.fits[4]...
Rescale intensity percentiles: 110.161376953, 173.693756409
Length of PHT result: 42
min(x0)=   1, min(x1)= 269, min(y0)= 827, min(y1)= 780
max(x0)=3852, max(x1)=4094, max(y0)=1611, max(y1)=1545
buf=200
topx=3896, topy=1848
...
Trail Direction: Right to Left
42 trail segment(s) detected
...
End point list:
    1. (1256, 1345), (2770, 1037)
    2. (  11, 1598), ( 269, 1545)
...
Total run time: 22.4326269627 s
>>> results[('jc8m10syq_flc.fits', 4)]
array([[[1242, 1348],
        [2840, 1023]],
       [[1272, 1341],
```

```
        [2688, 1053]],
       ...
       [[2697, 1055],
        [2967, 1000]]])
>>> errors
{}
```

Find trail segments for multiple images and all ACS/WFC science extensions with multiprocessing:

```
>>> results, errors = detsat(
...     '*_flc.fits', chips=[1, 4], n_processes=12, verbose=True)
6 file(s) found...
Using 12 processes
Number of trail segment(s) found:
  abell2744-hffpar_acs-wfc_f814w_13495_11_01_jc8n11q9q_flc.fits[1]: 0
  abell2744-hffpar_acs-wfc_f814w_13495_11_01_jc8n11q9q_flc.fits[4]: 4
  abell2744_acs-wfc_f814w_13495_51_04_jc8n51hoq_flc.fits[1]: 2
  abell2744_acs-wfc_f814w_13495_51_04_jc8n51hoq_flc.fits[4]: 34
  abell2744_acs-wfc_f814w_13495_93_02_jc8n93a7q_flc.fits[1]: 20
  abell2744_acs-wfc_f814w_13495_93_02_jc8n93a7q_flc.fits[4]: 20
  j8oc01sxq_flc.fits[1]: 0
  j8oc01sxq_flc.fits[4]: 0
  jc8m10syq_flc.fits[1]: 0
  jc8m10syq_flc.fits[4]: 38
  jc8m32j5q_flc.fits[1]: 42
  jc8m32j5q_flc.fits[4]: 12
Total run time: 34.6021330357 s
>>> results[('jc8m10syq_flc.fits', 4)]
array([[[1242, 1348],
        [2840, 1023]],
       [[1272, 1341],
        [2688, 1053]],
       ...
       [[2697, 1055],
        [2967, 1000]]])
>>> errors
{}
```

For a given image and extension, create a DQ mask for a satellite trail using the first segment (other segments should give similar masks) based on the results from above (plots not shown):

```
>>> trail_coords = results[('jc8m10syq_flc.fits', 4)]
>>> trail_segment = trail_coords[0]
>>> trail_segment
array([[1199, 1357],
       [2841, 1023]])
>>> mask = make_mask('jc8m10syq_flc.fits', 4, trail_segment,
...                  plot=True, verbose=True)
Rotation: -11.4976988695
Hit image edge at counter=26
Hit rotate edge at counter=38
Run time: 19.476323843 s
```

Update the corresponding DQ array using the mask from above:

```
>>> update_dq('jc8m10syq_flc.fits', 6, mask, verbose=True)
DQ flag value is 16384
Input... flagged NPIX=156362
Existing flagged NPIX=0
```

```
Newly... flagged NPIX=156362
jc8m10syq_flc.fits[6] updated
```

acstools.satdet.**detsat**(*searchpattern, chips=[1, 4], n_processes=4, sigma=2.0, low_thresh=0.1, h_thresh=0.5, small_edge=60, line_len=200, line_gap=75, percentile=(4.5, 93.0), buf=200, plot=False, verbose=True*)

Find satellite trails in the given images and extensions. The trails are calculated using Probabilistic Hough Transform.

---

**Note:** The trail endpoints found here are crude approximations. Use *make_mask* to create the actual DQ mask for the trail(s) of interest.

---

**Parameters**

**searchpattern** : str

Search pattern for input FITS images, as accepted by `glob.glob`.

**chips** : list

List of extensions for science data, as accepted by `astropy.io.fits`. The default values of `[1, 4]` are tailored for ACS/WFC.

**n_processes** : int

Number of processes for multiprocessing, which is only useful if you are processing a lot of images or extensions. If 1 is given, no multiprocessing is done.

**sigma** : float, optional

The size of a Gaussian filter to use before edge detection. The default is 2, which is good for almost all images.

**low_thresh** : float, optional

The lower threshold for hysteresis linking of edge pieces. This should be between 0 and 1, and less than `h_thresh`.

**h_thresh** : float, optional

The upper threshold for hysteresis linking of edge pieces. This should be between 0 and 1, and greater than `low_thresh`.

**small_edge** : int, optional

Size of perimeter of small objects to remove in edge image. This significantly reduces noise before doing Hough Transform. If it is set too high, you will remove the edge of the satellite you are trying to find.

**line_len** : int, optional

Minimum line length for Probabilistic Hough Transform to fit.

**line_gap** : int, optional

The largest gap in points allowed for the Probabilistic Hough Transform.

**percentile** : tuple of float, optional

The percent boundaries to scale the image to before creating edge image.

**buf** : int, optional

How close to the edge of the image the satellite trail has to be to be considered a trail.

**plot** : bool, optional

Make plots of edge image, Hough space transformation, and rescaled image. This is only applicable if `n_processes=1`.

**verbose** : bool, optional

Print extra information to the terminal, mostly for debugging. In multiprocessing mode, info from individual process is not printed.

**Returns**

**results** : dict

Dictionary mapping `(filename, ext)` to an array of endpoints of line segments in the format of `[[x0, y0], [x1, y1]]` (if found) or an empty array (if not). These are the segments that have been identified as making up part of a satellite trail.

**errors** : dict

Dictionary mapping `(filename, ext)` to the error message explaining why processing failed.

`acstools.satdet.`**`make_mask`**(*filename*, *ext*, *trail_coords*, *sublen=75*, *subwidth=200*, *order=3*, *sigma=4*, *pad=10*, *plot=False*, *verbose=False*)

Create DQ mask for an image for a given satellite trail. This mask can be added to existing DQ data using *update_dq*.

**Note:** Unlike *detsat*, multiprocessing is not available for this function.

**Parameters**

**filename** : str

FITS image filename.

**ext** : int, str, or tuple

Extension for science data, as accepted by `astropy.io.fits`.

**trail_coords** : ndarray

One of the trails returned by *detsat*. This must be in the format of `[[x0, y0], [x1, y1]]`.

**sublen** : int, optional

Length of strip to use as the fitting window for the trail.

**subwidth** : int, optional

Width of box to fit trail on.

**order** : int, optional

The order of the spline interpolation for image rotation. See `skimage.transform.rotate`.

**sigma** : float, optional

Sigma of the satellite trail for detection. If points are a given sigma above the background in the subregion then it is marked as a satellite. This may need to be lowered for resolved trails.

**pad** : int, optional

Amount of extra padding in pixels to give the satellite mask.

**plot** : bool, optional

Plot the result.

**verbose** : bool, optional

Print extra information to the terminal, mostly for debugging.

**Returns**

**mask** : ndarray

Boolean array marking the satellite trail with *True*.

**Raises**

**IndexError**

Invalid subarray indices.

**ValueError**

Image has no positive values, trail subarray too small, or trail profile not found.

acstools.satdet.**update_dq**(*filename*, *ext*, *mask*, *dqval=16384*, *verbose=True*)

Update the given image and DQ extension with the given satellite trails mask and flag.

**Parameters**

**filename** : str

FITS image filename to update.

**ext** : int, str, or tuple

DQ extension, as accepted by `astropy.io.fits`, to update.

**mask** : ndarray

Boolean mask, with *True* marking the satellite trail(s). This can be the result(s) from *make_mask*.

**dqval** : int, optional

DQ value to use for the trail. Default value of 16384 is tailored for ACS/WFC.

**verbose** : bool, optional

Print extra information to the terminal.

## 10.2 Version

acstools.satdet.**__version__ = u'0.3'**

unicode(object='') -> unicode object unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

acstools.satdet.**__vdate__ = u'07-Dec-2015'**

unicode(object='') -> unicode object unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

## 10.3 Author

acstools.satdet.**__author__ = u'David Borncamp, Pey Lian Lim'**

unicode(object='') -> unicode object unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

CHAPTER

ELEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

## a

## Symbols



## A



## C



## D



## M



## S



## U