



Instrument Science Report WFC3 2008-002

# Python Quick-Look Utilities for Ground WFC3 Images

---

A.R. Martel  
January 25, 2008

---

## ABSTRACT

*A Python module to process and manipulate ground WFC3 UVIS and IR images is presented. Several methods are defined to assemble and mosaic the images, calculate basic statistical quantities on the imaging areas and overscan regions, subtract the overscan and reference regions, trim the images, and calculate the UVIS dark current rate and IR noise. The module can be easily incorporated into Python scripts for more complex processing.*

---

## 1. Introduction

A simple Python module of quick-look utilities to process WFC3 UVIS and IR ground images is presented. The module is meant to provide a quick and easy way for any user to manipulate and analyze the ground images. The code can be easily customized to add new methods or modified for other instruments and detectors. It makes use of the latest versions of ‘numpy’, ‘pyfits’, and ‘pylab’, all already installed on the Institute machines as well as publicly available for personal desktops or laptops. No installation of software is required. Currently, the Python module treats the OPUS raw images located on the group partition of the WFC3 team. These consist of all the WFC3 images obtained in the ground calibration campaigns of WFC3 with the spare and flight detectors (~47000 images at the writing of this report). The module is structured around two classes, one for the UVIS channel and the other for the IR. Methods are defined in each class to perform simple operations, such as unscrambling an image and its overscan areas, calculating

Operated by the Association of Universities for Research in Astronomy, Inc., for the National Aeronautics and Space Administration

simple statistics on the different frame regions, fitting and subtracting the overscan areas from the imaging areas, and writing the results to disk. The classes also assign header keywords as attributes to the image object for easy access. When imported in a Python script, this module can be a powerful tool to process a large number of images. Similar routines, written in IDL, were presented by Hilbert (2004). In the following, I show how to use this module from the Python command line.

## 2. Header Attributes

The Python module is called `WFC3_Utills.py`. Change directory to where your OPUS `*_raw.fits` images reside and from the shell prompt, start Python, import the module `WFC3_Utills`, and create an image object with the function `'WFC3'` defined in the module. From the image header, `'WFC3'` will automatically detect if the image has the UVIS or IR format, and will accordingly instantiate the appropriate class, `'WFC3_UVIS'` or `'WFC3_IR'`. When invoking `'WFC3'`, some basic header keywords are printed, such as ID number, detector, amplifiers, gain, image type, exposure time, filter, subarray (“True” or “False”), and for the IR, the sampling sequence and the number of samples.

```
[73]tib:/home/martel/data% python
Python 2.5.1 (r251:54863, Jun 29 2007, 15:29:55) [C] on sunos5
Type "help", "copyright", "credits" or "license" for more information.
>>> from WFC3_Utills import *          # Assuming WFC3_Utills is in
your Python path, otherwise give the full path name.
```

```
WFC3_Utills          : Version 1.0, Jan 2008
PyFITS version       : 1.2
NumPy version        : 1.0.4
```

```
>>> image_uvis = WFC3('icsa0302r_07355142055_raw.fits')
45109 : UVIS-1 ABCD 1.5 FLAT 1.0 F814W False
>>> image_ir = WFC3('iaaabbccr_07177233935_raw.fits')
31421 : IR-1 ABCD 2.5 EXT 5.864582 Blank False RAPID 3
```

You can now manipulate `'image_uvis'` and `'image_ir'`. For example, all the header keywords from the primary header and the science headers, can be accessed as attributes (in small and large caps) :

```
>>> image_uvis.ccdamp
'ABCD'
>>> image_uvis.CCDAMP
'ABCD'
>>> image_ir.bitpix
16
>>> image_ir.tvnum
31421
>>> image_uvis.oslamp
'UNDEFINED'
>>> image_ir.obsmode
'MULTIACCUM'
```

More rigorously, the keywords can also be accessed via Python dictionaries :

```
>>> image_uvis.hdr['exptime'] # For the primary header.
1.0
>>> image_ir.hdr['obsmode']
'MULTIACCUM'
>>> image_uvis.hdrsci1['datamax'] # For the science extensions (two for
UVIS and NSAMP for IR).
33591.0
>>> image_uvis.hdrsci2['DATAMAX']
33484.0
>>> image_uvis.hdrsci2['naxis1']
4206
>>> image_ir.hdrsci3['bzero']
32768.0
```

and so on. The science data (sci1 and sci2 for UVIS and sci1, sci2, sci3, ... for IR) are also accessible as arrays :

```
>>> image_uvis.sci2
array([[ 2475.,  2499.,  2515., ...,  2514.,  2496.,  2465.],
       [ 2476.,  2498.,  2518., ...,  2514.,  2495.,  2463.],
       [ 2477.,  2502.,  2524., ...,  2511.,  2496.,  2464.],
       ...,
       [ 2474.,  2498.,  2519., ...,  2508.,  2487.,  2462.],
       [ 2471.,  2493.,  2520., ...,  2511.,  2491.,  2460.],
       [ 2469.,  2494.,  2513., ...,  2506.,  2490.,  2455.]],
      dtype=float32)

>>> image_ir.sci3
array([[ 8581., 14091., 15644., ..., 14353., 13165., 8090.],
       [ 9086., 14891., 14464., ..., 13243., 13123., 8259.],
       [10557., 14170., 14805., ..., 13786., 13405., 9562.],
       ...,
       [ 5694., 10519., 10077., ..., 13588., 12969., 9562.],
       [ 5800., 10678., 9640., ..., 12862., 12723., 8254.],
       [ 5295., 10438., 13014., ..., 14668., 12477., 8296.]],
      dtype=float32)
```

### 3. UVIS Methods

There are currently four methods that operate on the UVIS images :

1. *Unscramble* : Unscrambles the raw UVIS image into a full mosaic. The overscan areas are not subtracted and are not trimmed. The orientation is amps A-B at the top and C-D at the bottom as viewed with DS9. Subarrays remain the same but the orientation is also amp A in the top-left corner and amp D in the bottom-right corner. The unscrambled image is written as id(tvnum).fits by default where 'tvnum' is the ID number of the image in the database, but the user can provide any name.

```
>>> image_uvis.Unscramble()
```

```

Writing the unscrambled full-frame image or subarray to disk :
id45109.fits
>>> image_uvis.Unscramble(output="anothername.fits")
Writing the unscrambled full-frame image or subarray to disk :
anothername.fits
>>> image_uvis.Unscramble("somename.fits")
Writing the unscrambled full-frame image or subarray to disk :
somename.fits

```

2. *Regions* : This method defines all the imaging and overscan regions and optionally calculates statistics on those regions. The text file containing the results is written to disk as id(tvnum).dat by default. No overscans are currently defined for subarrays. The following read-out patterns and binning are implemented : ABCD (1x1), ABCD (2x2), ABCD (3x3), AD (1x1), AC (1x1), BC (1x1), BD (1x1). The regions are labeled as : IA (Imaging Area), SV (Serial Virtual), SP (Serial Physical), PV (Parallel Virtual). For each region of the image, the following statistical quantities are calculated : low value ("low"), high value ("high"), mean ("mean"), median("median"), and RMS noise ("rms\_noise"). All these quantities are accessible for each quadrant.

```

>>> image_uvis.Regions()
Calculating and writing the image statistics to file : id45109.dat
>>> image_uvis.mean['IA']['C']      # Imaging area, amp C
13699.7548828125
>>> image_uvis.mean['PV']['D']      # Parallel Virtual overscan, amp D
2605.6708470394738
>>> image_uvis.rms_noise['SV']['C'] # Serial Virtual overscan, amp C
2.0233799988813388
>>> image_uvis.median['IA']['B']    # Imaging area, amp B
14739.0
>>> image_uvis.Regions(stats=0)     # Define the regions but don't
calculate the statistics.

```

From the command line, you can loop through different quantities. For example, to get count rates in each of the imaging areas :

```

>>> for amp in image_uvis.ccdamp:
...     print amp, image_uvis.median['IA'][amp]/image_uvis.exptime
...
A 13278.0
B 14739.0
C 14099.0
D 15457.0

```

3. *Subtrim* : This is probably the most useful method. A linear fit to the median of the overscan regions specified by the user is subtracted row-by-row (for SV and SP) or column-by-column (for PV). By default, the serial virtual overscan (SV) is subtracted. No operation is performed on subarrays (except calculate their statistics if requested). The overscan levels are then subtracted from the imaging areas and the image is trimmed, mosaiced, and written to disk. Since this method calls the Regions method, all the statistical quantities are available (see above).

```

>>> image_uvis.Subtrim()
Calculating and writing the image statistics to file : id45109.dat

```

```

Writing the overscan-subtracted trimmed image to disk :
id45109_SV.fits
>>> image_uvis.Subtrim(overscan="PV")      # Subtract the PV overscan.
Calculating and writing the image statistics to file : id45109.dat
Writing the overscan-subtracted trimmed image to disk :
id45109_PV.fits
>>> image_uvis.Subtrim(overscan="SP", output="mosaic.fits")
# Subtract the SP overscan and rename the output.
Calculating and writing the image statistics to file : id45109.dat
Writing the overscan-subtracted trimmed image to disk : mosaic.fits
>>> image_uvis.Subtrim(overscan="NO", output="nooverscan.fits")
# Don't subtract the overscan.
Calculating and writing the image statistics to file : id45109.dat
Writing the overscan-subtracted trimmed image to disk :
nooverscan.fits

```

4. *Dark* : This method calculates the dark current (in DNs) of each quadrant of a UVIS dark frame. It is optimized for dark frames when the flight detector (UVIS1') is at its optimal temperature (-83 C). But the rates for any other image can also be calculated, such as a bias frame, for the same default parameters as a dark. The dark current is modeled as the mode of the histogram of each quadrant with two methods : 1) a Gaussian fit and 2) a 2nd-order polynomial fit. Only the bins with number counts above a level specified by the user will be included in the fits. For example, "level = 0.01" (the default) means that only the bins with counts that are at levels greater than 1% of the counts in the peak bin of the histogram will be fit. For "level = 0.50" (50%), only the top half of the histogram will be fit. For a long dark (>100 sec, say) with the cold detector, the hot pixel tail is at a very low level with respect to the peak, hence the default level of 0.01 (1%) is reasonable. But at warmer temperatures, the level needs to be increased to remain above the hot pixel tail. The options for this method are : a) "overscan = SV" : The bias level is subtracted using the specified overscan region (not a full frame bias subtraction). The choices are : SV = Serial Virtual Overscan (the default), SP = Serial Physical Overscan, PV = Parallel Virtual Overscan. b) "level = 0.01" : It is usually necessary to execute the method twice with different levels. As mentioned above, for a cold detector, a very low level of 0.01 (1%) gives an excellent Gaussian fit. But a higher level may be necessary to fit the peak with a 2nd-order polynomial (30-50% seems to work well). c) "clip = 0" : The hot pixel tail can be clipped from the histogram before doing the fits. For a cold detector, clipping seems unnecessary since the hot pixel tail is well below and away from the mode of the histogram ( $3\sigma$  or more away). For a warm detector (or a cold detector with extensive radiation damage), clipping should be used with caution; too many pixels could be clipped out of the histogram. By default, the clipping is performed three times at a  $3\sigma$  level from the median. d) "bin = 1" : The histogram can be binned by a factor of 1 (the default), 2, or 4. There are generally enough counts in bin widths of 1 for accurate fits. e) "plotdark = 1" : Generate a Postscript figure of the histogram. The 2nd order fit is superimposed as red circles and the Gaussian fit as a dashed line atop the histogram. The dark current rate of the UVIS2 detector in the second thermal vacuum campaign of WFC3 was calculated with this method (Martel 2007).

```
>>> image_uvvis.Dark() # the default
>>> image_uvvis.Dark(overscan = 'SV', level = 0.01, clip = 0, bin = 1,
plotdark = 1) # the explicit default
```

## 4. IR Methods

There are currently three methods defined in the IR class. Please note that the amplifiers are labeled as A, B, C, D as for the UVIS detector, as they are defined in the image header. But formally, they should be labeled as 1, 2, 3, and 4 where A=1, C=2, D=3, and B=4.

1. *Regions* : This method defines all the imaging and overscan regions and optionally calculates statistics on those regions. The text file containing the results for all extensions is written to disk as id(tvnum).dat by default. Full frames and subarrays in the standard four amp readout are implemented : binning of 1x1, SQ64Sub, SQ128Sub, SQ256Sub, and SQ512Sub. The regions are labeled as : IA (Imaging Area), HR (Horizontal Reference Pixels), and VR (Vertical Reference Pixels). For each region of the image, the following statistical quantities are calculated : low value (“low”), high value (“high”), mean (“mean”), median (“median”), and RMS noise (“rms\_noise”). The statistics on each region for each extension of the ramp are accessible.

```
>>> image_ir.Regions(stats=0) # Define the regions but don't
calculate the statistics.
>>> image_ir.Regions() # same as stats=1, the default
Calculating and writing the image statistics to file : id31421.dat
>>> image_ir.median[1]['HR']['D'] # for extension 1
12877.5
>>> image_ir.rms_noise[3]['VR']['B'] # for extension 3
662.74589869175475
>>> image_ir.high[2]['IA']['C'] # for extension 2
50654.0
```

2. *Subtrim* : This method calculates and subtracts the median of the reference pixel values from the corresponding image areas for full frame images and subarrays. This operation is performed for each extension of the image ramp sequence. By default, the median of the Vertical Reference pixels (VR) is removed. Also, by default, the 0th-image is subtracted from the last image of the sequence (“sub0th=1”), thus mimicking a “bias” subtraction. This image is then trimmed and written to disk.

```
>>> image_ir.Subtrim()
Calculating and writing the image statistics to file : id31421.dat
Writing the reference-subtracted trimmed image to disk :
id31421_VR.fits
>>> image_ir.Subtrim(refpix = 'HR', sub0th = 0,
output='nosub.fits')
Calculating and writing the image statistics to file : id31421.dat
Writing the reference-subtracted trimmed image to disk :
nosub.fits
```

3. *Noise* : Consecutive science extensions in the IR ramp image are subtracted pairwise and an image of the standard deviations of the residuals is created. A Gaussian is then fit to the standard deviations : the readnoise is the location of the peak and  $\sigma$  the uncertainty. The values are in counts (DNs). The peak is also modeled with a 2nd-order polynomial. In both cases (and similar to the UVIS *Dark* method described above), the histogram is fit above a certain fractional level from its peak, as specified by the user. The default is “level = 0.80” i.e. only the bins in the upper 20% portion of the distribution are included in the fit. A “level = 0.50” means that the top half of the histogram is fit. The reference pixel area (horizontal or vertical) to subtract from each quadrant before doing the pairwise subtraction can be specified (VR is the default). To minimize the contribution from the dark current, a RAPID sequence should be used; it will yield the most accurate noise measurements (see Hilbert 2005). The noise distribution and the fits can be plotted with the “plotnoise” option (“plotnoise = 1” by default : data as solid black line, Gaussian as dashed black line, and polynomial as solid colored line). Note that the binning of the histogram is optimized for the on-orbit operating temperature of the flight device FPA165 (IR4) (145 K or -128 C). For higher temperatures, the histogram will shift to the right (more noise) and the binning may need to be refined.

```
>>> image_ir.Noise() # the default
>>> image_ir.Noise(refpix = 'VR', level = 0.80, plotnoise = 1)
# the explicit default
>>> image_ir.Noise(refpix = 'HR', level = 0.20, plotnoise = 0)
# other options
```

## 5. Conclusion

A Python module to process and manipulate WFC3 UVIS and IR ground images was presented. Examples of its use were presented from the Python command line. A description and help on each method can be printed with the built-in function ‘help’ in Python : >>> help(WFC3), >>> help(WFC3\_UVIS.Subtrim), etc... Improvements in the code will be included in future releases. The latest version of this Python module is available by contacting the author by e-mail.

## References

Hilbert, B., ISR WFC3 2004-10, Basic IDL Data Reduction Algorithm for WFC3 IR and UVIS Channel

Hilbert, B., ISR WFC3 2005-15, Results of WFC3 Thermal Vacuum Testing : IR Channel Readnoise

Martel, A.R., ISR WFC3 2007-26, WFC3 TV2 Testing: UVIS-2 Dark Frames and Rates