CHAPTER 1:

GETTING STARTED WITH MULTIDRIZZLE

In this chapter. . .

MultiDrizzle can process a set of ACS observations to generate a distortion-free, cosmic-ray clean final image. The case presented in this chapter only represents one fairly common situation for ACS users which requires the use of most of MultiDrizzle's functionality.

This chapter delves right into the usage of MultiDrizzle with modest explanations along the way. More detailed descriptions of the parameters and syntax for advanced usage of MultiDrizzle can then be found in the next chapter.

1.1Starting Up...

Operation of **MultiDrizzle** can be run from Python or from **PyRAF**, along with **STSDAS 3.2**. MultiDrizzle has been designed as a Python package with it's own interface, however, it can also be loaded into **STSDAS** under **PyRAF** to enable use of the graphical EPAR interface that comes with **PyRAF**. These packages have been designed to work together in an almost seamless manner without requiring any manual setting of the environment by the user. **MultiDrizzle** requires **Python 2.3.3** (or later) and **numarray 1.2** (or later), and **F2PY** to be installed for operation with the Python interfaces, in addition, **STSDAS 3.2** and **PyRAF** 1.1 are required for use of the GUI EPAR interface.

1 Running under PyRAF

This example will demonstrate the use of the **MultiDrizzle** while running under **PyRAF**.

PyRAF can usually be started by simply running the command:

```
xyz> pyraf
```

Once **PyRAF** has started up, load the **STSDAS**, then **dither** packages:

```
--> stsdas
--> dither
```

At this point, the **PyDrizzle** code has been loaded into **PyRAF** for use. **MultiDrizzle** has been incorporated into **STSDAS** not only as a Python task in its own right, but also as an **IRAF** task with the usual parameter driven interface. This **IRAF** interface will be used to process the first dataset with **MultiDrizzle**.

The Python syntax can also be used from PyRAF, since PyRAF provides full Python functionality as well as access to IRAF tasks. This capability will be used for this example by demonstrating how to run MultiDrizzle under PyRAF.

2 Running under Python

MultiDrizzle can also be used directly from Python without loading IRAF, STSDAS or PyRAF. MultiDrizzle has been written as a Python package and can be treated just like any other Python package. Therefore, to use MultiDrizzle directly from Python, go to the directory with the data and start up the Python, usually just with:

```
xyz> python
```

The use of MultiDrizzle using the Python interface requires that the MultiDrizzle code be installed in a directory accessible by Python. If that directory is not already included in the PYTHONPATH or default Python system path, you can point to it using:

```
>>> sys.path.insert(1,'/directory/with/MultiDrizzle/code')
```

MultiDrizzle can then be loaded using the usual import mechanism in Python:

```
>>> import multidrizzle
```

At this point, MultiDrizzle can be run on the calibrated images in the local directory. Interactive help can be obtained for MultiDrizzle using the built-in help mechanism:

```
>>> multidrizzle.help()
```

This will provide basic information on the methods available for use and their syntax, including the method used for bringing up the Python GUI interface for editing all the parameters.

1.2First Session...

All ACS data processed by the standard calibration pipeline has been processed by **MultiDrizzle** to remove geometric distortion, and to remove cosmic rays when combining associated images. Data taken using a standard dither pattern, using CR-SPLIT on a long exposure or using REPEAT-OBS, as specificed in the observing proposal will produce associated data that will be automatically combined into a dither product by **MultiDrizzle**. There will be times, though, when the results are not sufficient for scientific analysis and the data requires reprocessing offline using **MultiDrizzle**. This example illustrates how to take observations associated by the HST calibration pipeline and reprocess them with **MultiDrizzle**.

The test data

This example will be based on observations taken using a 2-point Dither-Line pattern with CR-SPLIT=2 at each pointing using the WFC detector of the ACS. The association table, seen in Table Number range Table , will serve as the input to **MultiDrizzle**.

Association table for dithered observations

MEMNAME	MEMTYPE	MEMPRSNT
J8CW03A1Q	EXP-CR1	yes
J8CW03A2Q	EXP-CR1	yes
J8CW03FFQ	EXP-CR2	yes
J8CW03FJQ	EXP-CR2	yes
J8CW030X0	PROD-DTH	no
J8CW03021	PROD-CR1	yes
J8CW03031	PROD-CR2	yes

This example relies on data taken as part of the ACS ERO program. The association table, however, was manually created for this example to represent how the images could have been associated if they would have been taken as part of a formal Dither-pattern in the proposal.

It is assumed, though, that this association has already been processed using **CALACS.** The products created by CALACS include both cosmic-ray cleaned, calibrated products with 'crj.fits' suffixes and fully calibrated input images with 'flt.fits' suffixes. This will result in having at least the following files available in the current directory:

```
j8cw030x0_asn.fits
j8cw03021_crj.fits
j8cw03031_crj.fits
j8cw03a1q_flt.fits
j8cw03a2q_flt.fits
j8cw03ffq_flt.fits
j8cw03fjq_flt.fits
```

Reference files: IDCTAB and MDRIZTAB

These files will contain a reference to the distortion coefficients reference file, `IDCTAB`, and MultiDrizzle parameter table, MDRIZTAB, in their primary headers. The MultiDrizzle parameter table provides all the settings used by the pipeline to run MultiDrizzle on any given ACS single image or association. For these images, we find that `IDCTAB` and `MDRIZTAB` is:

```
--> hedit j8cw03a1q_flt.fits[0] idctab,mdriztab .

j8cw03a1q_flt.fits[0],IDCTAB = jref$wfc_smov_idc.fits

j8cw03a1q_flt.fits[0],MDRIZTAB = jref$wfc_smov_mdz.fits
```

The environment variable *jref* should be defined (at least within **PyRAF**) to point to the directory where the reference file is located. Simply use:

```
--> show jref
```

to verify this variable points to the correct directory. If not, it can be set using the standard **IRAF** syntax, where the directory shown in this example would be replaced by the correct one for your local system. It might, for example, be located at:

```
--> set jref = '/data/cdbs7/jref'
```

Running MultiDrizzle

The Python syntax for running MultiDrizzle does not vary that much from the PyRAF syntax. Unlike the PyRAF synt, it provides the user with

the ability to examine the inputs and computed parameters before spending the majority of time actually doing the image combination.

MultiDrizzle creates an instance of a MultiDrizzle object which contains all the methods necessary for editing the input parameters, computing the image combination parameters, then performing the actual cosmic-ray detection and image combination. All these functions get wrapped into one call through the PyRAF interface, making for simpler operation at the cost of the ability to inspect the object and its contents.

The processing starts with the creation of a MultiDrizzle object, which for the sake of this example will simply be called 'md':

```
-->  md = multidrizzle.Multidrizzle('j8cw030x0_asn.fits',mdriztab=yes)
```

The specification of the 'mdriztab' parameter in the initialization demonstrates how any of the input parameters whose values need to be changed from the default value can be provided at the start. If many parameters need to be edited, the Python GUI interface can now be started using the method:

```
--> md.editpars()
```

At this point, the Multidrizzle object has been provided all the input parameter values necessary for the processing. The computations of the parameters necessary for doing the image combination and cosmic-ray detection now need to be performed, using the method:

```
--> md.build()
```

The computed values can now be examined using usual Python introspection techniques, but the Multidrizzle object itself contains everything necessary for the processing. This can be started using:

```
--> md.run()
```

Session Summary

Despite all the explanations, very few commands are needed in order to process images or sets of images using MultiDrizzle. The total session described here simply boils down to:

```
xyz> cd /directory/with/data/

xyz> pyraf

--> stsdas

--> dither

--> multidrizzle j8cw030x0_asn.fits mdriztab=yes mode=h
```

The Python syntax does not require many more commnds:

```
--> import multidrizzle

--> md = multidrizzle.Multidrizzle('j8cw030x0_asn.fits',mdriztab=yes)

--> md.editpars()

--> md.build()

--> md.run()
```

Both ways of running MultiDrizzle will produce the exact same output products using the same input _flt.fits files.

1.3MultiDrizzle Syntax

Operation of MultiDrizzle can be performed using either (or both) an IRAF parameter-based interface or a native Python command-line interface. The IRAF provides the capability to use PyRAF's graphical EPAR editor to set all the parameters necessary for running MultiDrizzle, then execute it all the way to completion in one step. This method makes it easy to manage the input parameters through the graphical EPAR editor, but prevents introspection into or modification of the values and products computed by MultiDrizzle. This interface requires little explanation due to it's reliance of IRAF syntax, and will not be described here.

The native Python interface, on the other hand, utilizes the native Python syntax to provide the user with a simple method for running MultiDrizzle. Separate methods control computing the necessary parameters and the processing of the images using those parameters, while a separate method provides access to limited online help.

Getting Help

MultiDrizzle has a built-in help method accessible:

```
--> import multidrizzle
--> multidrizzle.help()
```

This will provide the most basic set of commands outlined in this example for running MultiDrizzle. Alternatively, a help method can provide a reminder of the available methods for any MultiDrizzle object already created using:

```
--> md = multidrizzle.Multidrizzle('flt.fits')
--> md.help()
```

where 'md' is an instance of a MultiDrizzle object created from all the 'flt.fits' files in the current directory.

Instantiating MultiDrizzle

MultiDrizzle can be executed directly from Python using the syntax:

```
--> md = multidrizzle.Multidrizzle(input='flt.fits',
output=None, editpars=False, **input_dict)
```

The parameter 'input_dict' serves as the means for the user to specify any of the remaining parameters from Table 2 which should have non-default values. Setting 'editpars' to True here will cause the Python GUI parameter editor to start automatically, rather than using the separate 'editpars()' method later. This editor allows the user to examine all the

inputs for MultiDrizzle and set them as necessary to work best with the input data.

Graphically Editing MultiDrizzle Parameters

MultiDrizzle relies on the **Traits** package from **SciPy** to manage the many input parameters, and provide a GUI interface for editing their values if desired. This GUI interface can be started using:

```
--> md.editpars()
```

The GUI interface will provide the ability to view and edit all input parameters used by **MultiDrizzle**, all 60-some parameters. Each parameter has been associated with a type of input, allowing MultiDrizzle to verify the inputs to some degree, or limiting the input choices to only valid values. The full set of parameters used by MultiDrizzle can be found in Table 2, along with any starting default values. Simply closing this GUI saves the values in the MultiDrizzle object for use.

Building Image Combination Parameters

Input parameters specify what the user wants to use for input, what processing needs to be performed, and what form the output image(s) should take. However, the actual processing gets performed using a myriad of routines; most notably, a callable version of the classic IRAF task 'drizzle'. The 'drizzle' task requires a fair number of inputs which must be derived from the input images and specification of the output frame. These values get computed using the method:

```
--> md.build()
```

This method passes the inputs to PyDrizzle to generate the necessary inputs for 'drizzle', and most importantly, create the internal list used to keep track of these values for all the MultiDrizzle processing steps.

Performing the Image Combination

The actual image processing can now begin using the method:

```
--> md.run(static=None, skysub=None, driz_separate=None, median=None,
blot=None, driz_cr=None, driz_combine=None)
```

This method allows the user one last chance to specify what processing steps should be performed. If no parameters are specified, all processing steps turned on during the instantiation of the MultiDrizzle object will be performed. Generally, though, the user simply wants to run this method without any specified parameters, such as:

```
--> md.run()
```

This represents the last step in processing images with MultiDrizzle, with the final product being a registered, cosmic-ray cleaned, distortion-free, photometrically uniform image.

Table Number range Table:Parameters for ***MultiDrizzle***

Parameter	***Default Value***	***Description***
input	flt.fits	Input files: filename, wildcard suffix, or @list
output		Rootname for output drizzled products
mdriztab	no	Use table with multidrizzle parameters?
refimage		Name of image to use as reference WCS
runfile	multidrizzle.run	File for logging the final drizzle commands
workinplace	no	Work on input files in place? (*NOT RECOMMENDED*)
coeffs	header	Use header-based distortion coefficients?
context	yes	Create context image during final drizzle?
clean	no	Remove temporary files?
group		Specification of extension/group to process
bits	0	Integer mask bit values considered good
ra		right ascension output frame center
dec		declination output frame center
build	yes	Create multi-extension output file?
shiftfile		Shiftfile name
Static Mask Creation		
static	yes	Create static bad-pixel mask from the data?
staticfile		Name of (optional) input static bad-pixel mask
static_sig	4	Sigma*rms below mode to clip for static mask
Sky Subtraction		
skysub	yes	Perform sky subtraction?
skywidth	0.1	Interval width for sky statistics (in sigma)
skystat	median	Sky correction statistics parameter
skylower	-50	Lower limit of usable data for sky (always in electrons)
skyupper	200	Upper limit of usable data for sky (always in electrons)
skyclip	5	Number of clipping iterations
skylsigma	4	Lower side clipping factor (in sigma)
skyusigma	4	Upper side clipping factor (in sigma)
skyuser		KEYWORD indicating a sky subtraction value if done by user.
Create Separate Drizzled Images		
driz_separate	yes	Drizzle onto separate output images?
driz_sep_outnx		Size of separate output frame's X-axis (pixels)
driz_sep_outny		Size of separate output frame's Y-axis (pixels)
driz_sep_kernel	turbo	Shape of kernel function
driz_sep_scale	INDEF	Absolute size of output pixels in arcsec/pixel

Parameter	*Default Value*	*Description*
driz_sep_pixfrac	1	Linear size of drop in input pixels
driz_sep_rot	INDEF	Orientation of final image's Y-axis w.r.t. North (in degrees)
driz_sep_fillval	INDEF	Value to be assigned to undefined output points
Create Median Image		
median	yes	Create a median image?
median_newmasks	yes	Create new masks when doing the median?
combine_type	median	Type of combine operation
combine_nsigma	6 3	Significance for accepting minimum instead of median
combine_nlow	0	*minmax*: Number of low pixels to reject
combine_nhigh	1	*minmax*: Number of high pixels to reject
combine_lthresh	INDEF	Lower threshold for clipping input pixel values
combine_hthresh	INDEF	Upper threshold for clipping input pixel values
combine_grow	1	Radius (pixels) for neighbor rejection
Blot Median Image		
blot	yes	Blot the median back to the input frame?
blot_interp	poly5	Interpolant: nearest,linear,poly3,poly5,sinc
blot_sinscl	1	Scale for sinc interpolation kernel
Remove Cosmic-rays		
driz_cr	yes	Perform CR rejection with deriv and driz_cr?
driz_cr_corr	no	Create CR cleaned _cor file and a _crmask file?
driz_cr_snr	3.0 2.5	*driz_cr:* SNR parameter
driz_cr_scale	1.2 0.7	*driz_cr:* scale parameter
driz_combine	yes	Perform final drizzle image combination?
Create Final Cleaned, Combined Product		
final_outnx		Size of FINAL output frame X-axis (pixels)
final_outny		Size of FINAL output frame Y-axis (pixels)
final_kernel	square	Shape of kernel function
final_scale	INDEF	Absolute size of output pixels in arcsec/pixel
final_pixfrac	1	Linear size of drop in input pixels
final_rot	0	Orientation of final image's Y-axis w.r.t. North (in degrees)
final_fillval	INDEF	Value to be assigned to undefined output points
Override Instrument Specific Parameters		
gain		Detector gain
gnkeyword		Detector gain keyword in header
rdnoise		Detector read noise
rnkeyword		Detector read noise keyword in header
exptime		Exposure time

Parameter	*Default Value*	*Description*
expkeyword		Exposure time keyword in header
crbit		Bit value for CR identification in DQ array

1.4MultiDrizzle Parameters

MultiDrizzle processing starts with a list of input images, then applies a number of operations on them to produce the final cosmic-ray cleaned image; specifically:

1. Initialization
2. Bad pixel (static) mask creation
3. Sky subtraction
4. Drizzle onto separate, registered output images
5. Combine the separate drizzled images into a median
6. "blot", or transform back the median to each input image
7. Compare the median with original images to make cosmic ray masks
8. Drizzle all the images onto a final image, using the cosmic ray masks

It relies on a large number of parameters for specifying what steps get performed and for controlling the algorithm used by each step. The complete list of parameters and their default values are given in Table 2. It is generally recommended to try running the script first with the default parameters, which should allow the task to process nearly any set of images for an initial review. The script can then be re-run, or restarted at a particular step, with modified parameters if this is necessary.

1 Initialization Step

Parameters: input, output, mdriztab, refimage, runfile, workinplace, coeffs, context, clean, group, bits, ra, dec, build, shiftfile

Processing: MultiDrizzle starts by determining what files are being specified as inputs. These files then get converted as necessary to a usable input format with appropriate units. This conversion includes making copies of each input file, if 'workinplace' is set to 'False', so that the original inputs SCI arrays are not altered and can be used for successive runs. A set of drizzle parameter values needed to combine the images into a final output image then get computed using PyDrizzle.

Input: This controls all inputs for the entire MultiDrizzle task, thus all the parameters are processed as input. In addition, it requires access to:

- the image files specified in the 'input' parameter
- the MDRIZTAB reference table specified in the input image headers

- the IDCTAB reference table specified in the input image headers
- any specified reference image as named in the 'refimage' parameter

Output: This step can result in the creation of several files, including:

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

Definitions: The definitions for each parameter are:

input: *[format: string]*

The name or names of the input files to be processed. It can be provided in any of several forms; namely,

filename of a single image
filename of an association (ASN)table
wild-card specification for files in directory
comma-separated list of filenames
'@file' filelist containing list of desired inputs filenames

The filelist needs to be provided as an ASCII text file containing a list of filenames for all input images with one filename on each line of the file. If inverse variance maps have also been created by the user and are to be used (by specifying 'IVM' to the parameter 'final_wht_type' described further below), then these are simply provided as a second column in the filelist, with each IVM filename listed on the same line as a second entry, after its corresponding exposure filename.

output: *'final' (default) [format: string]*

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

mdriztab: *[format: string]*

Specifies whether or not to use an MDRIZTAB reference table to specify the remaining MultiDrizzle parameter settings. If 'True', the values in the table will override the settings for the remainder of the parameters.

refimage: *[format: string]*

Optional "reference image" that can be provided, in which case MultiDrizzle will create a final product with the same WCS. This reference image should be a simple FITS file (single-group, no multiple extensions), and should have been already drizzled so that all its distortion has been removed, and its WCS is completely rectified.

runfile: *"multidrizzle.run" (default) [format: string]*
This log file will contain the IRAF CL commands necessary for performing the final combination manually using the "drizzle" task directly.

workinplace: *False (default), True [format: Boolean]*
This parameter specifies whether to perform all processing, including skysubtraction and update of the DQ array, on the original input or not. If set to 'True', then no copy of the input will be created for processing, and the original input will be modified directly by MultiDrizzle.

coeffs: *"header" (default) [format: string]*
The source of the distortion coefficients gets specified using:

header	use the 'header' for determining what distortion coefficients to use.
cubic	use 'cubic' solutions originally provided with 'drizzle'
trauger	use 'trauger' solutions originally provided with 'drizzle'
None	Do not apply any distortion correction to the images

Alternatively, an arbitrary distortion coefficients file can be specified (including optionally the full pathname). This distortion file is used in computing the WCS of the header.
NOTE: When the 'header' option has been selected, if the IDCTAB file is not found on disk, then the task will exit for ACS data, while for WFPC2 data it will default to the older "Trauger" model.

context: *False (default), True [format: Boolean]*
This parameter specifies whether or not to create a context image during the final drizzle combination. The context image contains the information on what image(s) contributed to each pixel encoded as a bit-mask. More information on context images can be obtained from the ACS Data Handbook online at:
http://www.stsci.edu/instruments/acs/

clean: *False (default), True [format: Boolean]*
The temporary files created by MultiDrizzle can be automatically removed by setting this parameter to 'True'. The affected files would include the coefficients files and static mask files created by PyDrizzle, along with other intermediate files created by MultiDrizzle. It is often useful to retain the intermediate files and examine them when first learning how to run MultiDrizzle. But when running it routinely, these files can be removed to save space.

group: *[format: string]*

A single FITS extension or group can be drizzled by setting this parameter. If a section is provided, then only that chip will be drizzled onto the output frame. Either a FITS extension number or GEIS group number (such as '1'), or a FITS extension name (such as 'sci,1') can be provided.

bits: *0 (default) [format: integer]*
Integer sum of all the DQ bit values from the input image's DQ array that should be considered 'good' when building the weighting mask. The default value of 0 indicates that all non-zero DQ bits values should be considered bad when computing the weighting for those pixels.

This value can be set by adding the integer values for all DQ bits that should be considered good. For example, the default value for ACS data combines the DQ values of 2 + 128 + 256 + 1024 + 2048 for a value of 3458. Thus, any ACS pixel which has only those DQ values would be considered as good pixels by MultiDrizzle. This can also be used to reset pixels to good if they had been flagged as cosmic rays during a previous run of MultiDrizzle, by adding the value 4096 for ACS and WFPC2 data.

ra: *[format: float]*
Right ascension (in decimal degrees) of the center of the output image. If this is not specified, the code will calculate the center automatically based on the distribution of image dither positions.

dec: *[format: float]*
Declination (in decimal degrees) of the center of the output image. If this is not specified, the code will calculate the center automatically based on the distribution of image dither positions.

build: *True (default) [format: Boolean]*
MultiDrizzle would combine the separate 'drizzle' output files into a single multi-extension format FITS file when this parameter gets set to 'True'. This combined output file will contain a SCI (science), a WHT (weight), and a CTX (context) extension. If set to 'False', each extension would remain as a separate simple FITS file on its own.

shiftfile: *[format: string]*
Name of optional input file containing the shifts to be applied to the input images to improve the registration of the images. These shifts will be added to those calculated automatically from the image headers. The specification for this file can be found at:
http://stsdas.stsci.edu/pydrizzle/tutorial/reports.html

gain: *[format: float]*
Value used to override instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the *gainkeyword* parameter is in use.

***gainkeyword**: [format: string]*
Keyword used to specify a value to be used to override instrument specific default gain values. The value is assumed to be in units of electrons/count. This parameter should not be populated if the *gain* parameter is in use.

rdnoise: [format: float]
Value used to override instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the *rnkeyword* parameter is in use.

***rnkeyword**: [format: string]*
Keyword used to specify a value to be used to override instrument specific default readnoise values. The value is assumed to be in units of electrons. This parameter should not be populated if the *rdnoise* parameter is in use.

***exptime**: [format: float]*
Value used to override default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the *expkeyword* parameter is in use.

***expkeyword**: [format: string]*
Keyword used to specify a value to be used to override default exposure time image header values. The value is assumed to be in units of seconds. This parameter should not be populated if the *exptime* parameter is in use.

***crbit**: [format: integer]*
Integer used to override instrument specific cosmic ray bit values. This value is used by Multidrizzle to update data quality arrays when cosmic rays or other image defects are identified as "bad" in the DRIZ_CR step. To prevent the image's data quality array from being updated set the *crbit* value to 0.

2 Bad Pixel (Static) Mask Creation

Parameters: static, staticfile, static_sig

Processing: A static mask gets created from all input images to flag pixels which are significantly below the mode. A separate static mask gets generated for each input chip that comprises a single exposure. For example, each ACS WFC image contains a separate image for each of 2 CCDs. A bad pixel gets defined as any pixel which falls more than 'static_sig' *RMS below the mode for a given chip or extension. Those

severely negative or low pixels can result from oversubtraction of bad pixels in the dark image during calibration.

The final static mask for each chip contains all the bad pixels that meet this criteria from all the input images. This static mask could also be combined with a user supplied static mask specified in the 'staticfile' parameter.

Input: Aside from the input parameters, this step requires opening each input image to access the science (SCI) extensions for generating the static masks.

Output: The generated static masks exist strictly in memory as numarray objects that get applied during the single drizzle step. They also get used to update the input mask files for the final image combination.

Definitions: The definitions for each parameter are:

static*: True (default), False [format: Boolean]*
This parameter specifies whether to create a static bad-pixel mask from the data or not. This mask flags all pixels that deviate by more than 'static_sig' sigma below the image median.

staticfile*: [format: string]*
Name of (optional) input static bad-pixel mask. This mask will be applied to all input images. If this is not specified, the code uses the bad pixel information associated with the images (eg the "dq" arrays for ACS, or the ".c1h" files for WFPC2) for the final drizzle combination.

static_sig*: 4.0 (default) [format: float]*
Number of sigma below the RMS to use as the clipping limit for creating the static mask.

3 Sky Subtraction

Parameters: skysub, skywidth, skystat, skylower, skyupper, skyclip, skylsigma, skyusigma, skyuser

Processing: The clipped mode gets computed for each input chip and scaled to a reference plate scale as an estimate of the sky background. The lowest scaled value for each chip of an observation (file) then gets re-scaled to the chip's plate scale and subtracted as the sky value. The primary header of each input image gets updated with this value.

In lieu of having MultiDrizzle computed the sky value, the user can supply their own sky value as a keyword in the input file header. This keyword name would then be given to MultiDrizzle in the 'skyuser' parameter.

Input: Aside from the input parameters, this step requires opening each input image to access the science (SCI) extensions for computing the sky values.

Output: The input file Primary headers get updated with the computed sky value, and each input image's SCI array (or copy, if 'workinplace' is set to False), gets sky subtracted.

Definitions: The definitions for each parameter are:

skysub*: True (default), False [format: boolean]*
Turn on or off sky subtraction on the input data.

skywidth*: 0.1 (default) [format: float]*
Bin width, in sigma, used to sample the distribution of pixel flux values in order to compute the sky background statistics.

skystat*: "median" (default) [format: string]*
Statistical method for determining the sky value from the image pixel values. Valid options are:

- median
- mode
- mean

skylower*: [format: float]*
Lower limit of usable pixel values for computing the sky. This value should be specified in units of electrons.

skyupper*: [format: float]*
Upper limit of usable pixel values for computing the sky. This value should be specified in units of electrons.

skyclip*: 5 (default) [format: integer]*
Number of clipping iterations to use when computing the sky value.

skylsigma*: 4.0 (default) [format: float]*
Lower clipping limit, in sigma, used when computing the sky value.

Skyusigma: *4.0 (default) [format: float]*
Upper clipping limit, in sigma, used when computing the sky value.

Skyuser: *[format: string]*
Name of header keyword which records the sky value already subtracted from the image by the user.

4 Drizzling to Separate Outputs

Parameters: driz_separate, driz_sep_outnx, driz_sep_outny, driz_sep_kernel, driz_sep_scale, driz_sep_pixfrac, driz_sep_rot, driz_sep_fillval

Processing: Each input image gets drizzled onto separate copies of the output frame. These copies when stacked would correspond to the final combined product. As separate image, though, they allow for treatment of each input image separately in the undistorted, final WCS system.

These images provide the information necessary for refining the image registration for the input image.

Input: Aside from the input parameters, this step requires:

- valid input images with SCI extensions
- valid distortion coefficients tables
- any optional secondary distortion correction images
- numarray object (in memory) for static mask

Output: This step produces:

- singly drizzled science image (simple FITS format)
- singly drizzled weight images (simple FITS format)

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

Definitions: The definitions for each parameter are:

driz_separate: True(default), False [format: boolean]
This parameter specifies whether or not to drizzle each input image onto separate output images. The separate output images will all have the same WCS as the final combined output frame. These images are used to create the median image, needed for the cosmic ray rejection step further on.

driz_sep_outnx: [format: float]
Size of the X axis of the output images, in pixels, which each input will be drizzled onto. If no value is specified, it will use the smallest size that can accommodate the full image.

driz_sep_outny: [format: float]
Size of the Y axis of the output images, in pixels, which each input will be drizzled onto. If no value is specified, it will use the smallest size that can accommodate the full image.

driz_sep_kernel:"turbo" (default) [format: string]

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

square	original classic drizzling kernel
point	the kernel is a point so each input pixelcan only contribute to the single pixel which is closest to the output position. It is equivalent to the limit pixfrac -> 0. It is very fast.
gaussian	the kernel is a circular gaussian with FWHM equal to the value of pixfrac, measured in input pixels.
turbo	this is similar to kernel="square" but the box is always the same shape and size on the output grid and always aligned with the X and Y axes. This results in a significant speed increase in some cases.
tophat	the kernel is a circular "top hat" shape of width pixfrac. In effect only output pixels within pixfrac/2 of the output position are affected.
lanczos3	a Lanczos style kernel extending 3 pixels from the center. The Lanczos kernel is a damped, bounded form of the "sinc" interpolator and is very effective for resampling single images when scale=pixfrac=1. It leads to less resolution loss than the other kernels, and also less correlated noise in outputs. It is however much slower. It should never be used for pixfrac != 1.0 and is not recommended for scale != 1.0.

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

driz_sep_scale*: None (default) [format: float]*

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

driz_sep_pixfrac*: 1 (default) [format: float]*

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

driz_sep_rot*: None (default) [format: float]*

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

driz_sep_fillval*: "INDEF" (default) [format: string]*
Value to be assigned to output pixels that have zero weight or did not receive flux from any input pixels during drizzling. This parameter corresponds to the 'fillval' parameter of the 'drizzle' task. If the default of 'INDEF' is used and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (eg. 0), then these pixels will be set to that value.

5 Median Image Creation

Parameters: median, median_newmasks, combine_type, combine_nsigma, combine_nlow, combine_nhigh, combine_lthresh, combine_hthresh, combine_grow

Processing: The combined median image gets created during this step by combining the singly drizzled science images. This median combination gets performed section-by-section from the input single drizzle images. Each section corresponds to a contiguous set of lines from each image taking up no more than 1Mb in memory, so that combining 10 input images would only require 10Mb for these sections.

Input: Aside from the input parameters, this step requires access to the single drizzled images on disk.

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

Definitions: The definitions for each parameter are:

median*: True (default), False [format: boolean]*
The user can specify whether or npt to create a median image with this parameter. This median image will be used as the comparison 'truth' image in the cosmic ray rejection step.

median_newmasks*: True (default), False [format: boolean]*
The user can specify whether or not to create new mask files when creating the median image. These masks are generated from the weight files produced previously by the "driz_separate" step, and would contain

all the bad pixel information. These pixels will be excluded when calculating the median. Generally this step should be set to "yes", unless it is desired to include bad pixels in generating the median.

***combine_type**: "minmed" (default) [format: string]*
This parameter allows the user to choose what method should be used to create the median image. Valid options are:

- average
- median
- sum
- minmed

The 'average', 'median', and 'sum' options set the mode of operation for using '*numcombine*', a numarray method for median-combining arrays, to create the median image. The "minmed" option will produce an image that is generally the same as the median, except in cases where the median is significantly higher than the minimum good pixel value, in which case it will choose the minimum. The sigma thresholds for this decision are provided by the "combine_nsigma" parameter.

***combine_nsigma**: "4 3"(default) [format: string]*
Sigmas used for accepting minimum values instead of median values when using the 'minmed' combination method. If two values are specified, then the first value will be used in the initial choice between median and minimum, while the second value will be used in the "growing" step to reject additional pixels around those identified in the first step. If only one value is specified, then it is used in both steps.

***combine_nlow**: [format: integer]*
When using a 'minmax' rejection method, this parameter sets the number of low value pixels to reject.

***combine_nhigh**: [format: integer]*
When using a 'minmax' rejection method, this parameters sets the number of high value pixels to reject.

***combine_lthresh**: None (default) [format: float]*
Sets the lower threshold for clipping input pixel values during image combination. This value gets passed directly to 'imcombine' for use in creating the median image. If None, no thresholds are used at all.

***combine_hthresh**: None (default) [format: float]*
Sets the upper threshold for clipping input pixel values during image combination. This value gets passed directly to 'imcombine' for use in creating the median image. If None, no thresholds are used at all.

***combine_grow**: 1 (default) [format: float]*

Width in pixels for additional pixels to be rejected in an image with a rejected pixel from one of the rejection algorithms. This parameter is used to set the 'grow' parameter in 'imcombine' for use in creating the median image.

6 Blotting Median Image

Parameters: blot, blot_interp, blot_sinscl

Processing: The median image has the distortion model applied to it to re-create 'cleaned' versions of the distorted input images; otherwise known as 'blotting' the median image. These blotted image can then be directly compared to the original distorted input images for detection of bad-pixels, hot pixels, and cosmic-rays for removal.

Input: Aside from the input parameters, this step only requires opening the single median image created from all the inputs images.

Output: A distorted version of the median image corresponding to each input 'chip' (extension) gets written out as output from this step as separate simple FITS images.

Definitions: The definitions for each parameter are:

blot*: True (default), False [format: boolean]*
Perform the blot operation on the median image? The output will be median smoothed images which match each input chip's image, and are used in the cosmic ray rejection step.

blot_interp*: "poly5" (default) [format: string]*
Type of interpolation to use when blotting drizzled images back to their original WCS. Valid options are:

nearest	Nearest neighbor
linear	Bilinear interpolation in x and y
poly3	Third order interior polynomial in x and y
poly5	Fifth order interior polynomial in x and y
sinc	Sinc interpolation; accurate but slow

The poly5 interpolation method has been chosen as the default because it is relatively fast and accurate. If 'sinc' interpolation has been selected, then it will use the value of the parameter 'blot_sinscl' to specify the size of the sinc interpolation kernel.

blot_sinscl*: 1.0 (default) [format: float]*

Size of the sinc interpolation kernel in pixels.

7 Cosmic-ray detection and removal

Parameters: driz_cr, driz_cr_corr, driz_cr_snr, driz_cr_scale

Processing: The blotted median images get compared to the original input images to detect any spurious pixels in each input. Those spurious pixels then get flagged as 'bad' in the mask files that get used as input for the final combination so that they do not show up in the final product.

Input: Aside from the input parameters, this step requires:

- the blotted median images, and
- the mask files.

Output: The identified bad pixels get flagged by updating the input mask files. Optionally, copies of the original images with the bad pixels removed can be created through the use of the 'driz_cr_corr' parameter.

Definitions: The definitions for each parameter are:

driz_cr*: True (default), False [format: boolean]*
Perform cosmic-ray detection? If set to "yes", it will detect cosmic-rays and create cosmic-ray masks using the algorithms from 'deriv' and 'driz_cr'.

d*riz_cr_corr**: False (default), True [format: boolean]*
Create a cosmic-ray cleaned input image? The cosmic-ray cleaned _cor image will be generated directly from the input image, and a corresponding _crmask file will be written to document the pixels detected as affected by cosmic-rays.

driz_cr_snr*: "3.5 3.0" (default) [format: string]*
These values specify the signal-to-noise ratios for the 'driz_cr' task to use in detecting cosmic rays. This parameter value gets passed directly to 'driz_cr'; see the help file for 'driz_cr' for further discussion of this parameter.

driz_cr_scale*: "1.2 0.7" (default) [format: string]*
Scaling factor applied to the derivative in 'driz_cr' when detecting cosmic-rays. This parameter gets passed directly to 'driz_cr'; see the help file for 'driz_cr' for further discussion of this parameter.

8 Final image combination

Parameters: driz_combine, final_wht_type, final_outnx, final_outny, final_kernel, final_scale, final_pixfrac, final_rot, final_fillval

Processing: This step performs the final image combination of the original input images (or their copies) using the updated mask files to remove any cosmic-rays. The output frame, just like the single drizzle step, can be redefined here using some parameters for this step, otherwise it will use the default computed to (ideally) include all the input pixels from all the input images after registering them according to their header's WCS information.

Input: Aside from the input parameters, this step requires:

- all input images SCI arrays
- In addition, there are many other cases such as this where enhanced error messages or error trapping can take place, and we feel that all these should be implemented in a coherent, consistent manner for the entire task.
  updated mask files
- all distortion coefficients files

Output: The final product of MultiDrizzle is a registered, cosmic-ray cleaned, distortion-free, photometrically flat science image with associated weight and context images. By default, these will be written out as a single multi-extension FITS file, but the user could simply have them written out as separate simple FITS images.

Definitions: The definitions for each parameter are:

driz_combine*: True (default), False [format: boolean]*
This parameter specifies whether or not to perform the final drizzle image combination. This applies the generated cosmic-ray masks to the input images and create a final, cleaned, distortion-corrected product.

final_wht_type*: EXP (default) [format: string]*
Specify the type of weighting image to combine with the bad pixel mask for the final drizzle step. The options are:

- EXP
- IVM
- ERR

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

Specifying 'ERR' is an alternative for ACS and STIS data, in which case the final drizzled images will be weighted according to the inverse variance of each pixel in the input exposure files, calculated from the error array data extension that is in each calibrated input exposure file. This array encapsulates all the noise sources in each exposure, including read-noise, dark current and sky background, as well as Poisson noise from the sources themselves, and this also includes a dependence upon exposure time. For WFPC2, the ERR array is not produced during calibration, therefore this option is not available. But for ACS and STIS datasets this option is generally recommended to be the most accurate type of weighting for producing the final drizzled image.

Finally, 'IVM' can be specified, in which case the user supplies their own inverse-variance weighting map. This may be necessary for specific purposes, for example to create a drizzled weight file for software such as Sextractor, which expects a weight image that contains all the background noise sources (sky level, read-noise, dark current, etc) but not the Poisson noise from the objects themselves. The user creates the inverse variance images and then specifies their names using the 'input' parameter for MultiDrizzle to specify an '@file'. This would be a single ASCII file containing the list of input calibrated exposure filenames (one per line), with a second column containing the name of the IVM file corresponding to each calibrated exposure. Each IVM file must have the same file format as the input file, and if given as multi-extension FITS files (for example, ACS or STIS data) then the IVM extension must have the EXTNAME of 'IVM'.

***final_outnx**: [format: float]*

Size of the X axis of the final drizzled image (in pixels). If no value is specified, it will use the smallest size that can accommodate the full image.

***final_outny**: [format: float]*

Size of the Y axis of the final drizzled image (in pixels). If no value is specified, it will use the smallest size that can accommodate the full image.

***final_kernel**: "square" (default) [format: string]*

Shape of the kernel used by 'drizzle' in the final image combination. The supported choices are:

- square
- point
- gaussian
- turbo
- tophat
- lanczos3

See the expanded descriptions of these kernels in the 'Drizzling to Separate Outputs' section.

***final_scale**: None (default) [format: float]*

Linear size of the output pixels in arcseconds/pixel for the final combined product. The default value of INDEF specifies that the undistorted pixel scale for the first input image, as computed by PyDrizzle, will be used as the pixel scale for the final output image.

***final_pixfrac**: 1. (default) [format: float]*

Fraction by which input pixels are "shrunk" before being drizzled onto the output image grid, given as a real number between 0 and 1. This specifies the size of the footprint, or "dropsize", of a pixel in units of the input pixel size. If pixfrac is set to less than 0.001, the kernel is reset to 'point' for more efficient processing. If more than a few images are being combined, values smaller than 1 (eg 0.7 or 0.8) can be specified, which result in a slightly sharper output image. For more information, read the help for the task 'drizzle'.

***final_rot**: 0. (default) [format: float]*

Position Angle of output image's Y-axis relative to North. The default of 0.0 would orient the final output image with North up. A value of INDEF would specify that the images will not be rotated, but will instead be drizzled in the default orientation for the camera, with the x and y axes of the drizzled image corresponding approximately to the detector axes.

***final_fillval**: "INDEF" (default) [format: string]*

Value to be assigned to output pixels that have zero weight or did not receive flux from any input pixels during drizzling. This parameter corresponds to the 'fillval' parameter of the 'drizzle' task. If the default of 'INDEF' is used and if the weight in both the input and output images for a given pixel are zero, then the output pixel will be set to the value it would have had if the input had a non-zero weight. Otherwise, if a numerical value is provided (eg. 0), then these pixels will be set to that value.