MIRI Simulators Documentation

Release 0.5devxx

MIRI Software Team

June 26, 2014

CONTENTS

Release 0.5devxx

Date June 26, 2014

CHAPTER ONE

INTRODUCTION

This document describes the implementation details for the MIRI general simulators package (simulators).

CHAPTER

TWO

MODULES

General purpose simulator modules may be found in the simulators/lib directory

2.1 File Searching Utility (`miri.simulators.find_simulator_file`)

2.1.1 Description

This module contains a utility which finds a named data file or configuration file which is stored somewhere within the miri.simulators package.

2.1.2 Objects

None.

2.1.3 Functions

`miri.simulators.find_simulator_file.`**`find_simulator_file`**(*filename*)

Find a file matching the given name first in a given search path of directories and (failing that) within the PYTHONPATH. By default, this function will search for a file within the current directory the MIRI module data directories and then try PYTHONPATH.

Parameters

filename: str The name of the file to be located.

Returns

filepath: str The full path and name of the matching file. If no file is found an empty string is returned.

2.2 Poisson integrator module (`miri.simulators.poisson_integrator`)

2.2.1 Description

This module simulates the behaviour of a detector made of electron counting pixels, when the incoming flux is small. It can be reused in any simulation where a particle integrator obeying Poisson statistics is needed.

The imperfect integrator adds zero-point and latency effects.

2.2.2 Objects

class `miri.simulators.poisson_integrator.`**`PoissonIntegrator`**(*rows*, *columns*, *particle='photon'*, *time_unit='seconds'*, *bucket_size=None*, *simulate_poisson_noise=True*, *verbose=1*)

Class PoissonIntegrator - Simulates the behaviour of a particle counting detector in which the actual number of particles (be they photons or electrons, etc...) detected varies with the expected number according to Poisson statistics.

This class simulates a perfect integrator.

Parameters

rows: int The number of detector rows. Must be at least 1.

columns: int The number of detector columns. Must be at least 1.

particle: string, optional, default="photon" The name of the particles being counted.

time_unit: string, optional, default="seconds" The unit of time measurement.

bucket_size: int, optional, default=None The maximum number of particles that may be counted. If None there is no limit.

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

Attributes

shape: tuple of 2 ints The shape of the integrator array.

particle: str The name of the particle/charge being counted (e.g. "photon" or "electron").

time_unit: str The name of the time unit in which integration durations are measured.

bucket_size: int The maximum number of paricles/charges that can be accumulated within any element. The element saturates when the bucket size is reached.

expected_count: int array The particles/charges accumulated within the elements of the integrator at the current time.

last_readout: int The reading obtained from the integrator when the expected_count was last sampled.

flux: float array The last flux on which the integrator was integrated.

nints: int The number of integrations executed since the PoissonIntegrator object was created, or since a new exposure was started. An integration is defined as the period between resets. An integration consists of one or more integration periods. The count may be sampled many times during an integration, and a graph of count against time typically looks like a ramp whose slope depends on the input flux.

nperiods: int The number of integration periods executed since the beginning of the integration. An integration period is a period of time during which the counter is integrated on a flux. Normally, the counter is read out at the end of the integration flagment.

readings: int The number of readings made since the beginning of the integration. Normally, this will be the same as the number of integration periods, but there is nothing to stop the counter being integrated more than once in between readings, or being read out more than once after an integration period. The latter strategy can be used to make a group of readings to reduce read noise.

nperiods_at_readout: int The number of integration periods executed before the last readout.

exposure: float The total exposure time for the current integration, i.e. the total accumulated integration time since the last reset.

`get_counts`()
Get the array containing the number of particles counted. This can be used to preserve a record of a previous exposure before an integrator object is deleted.

Returns

count: nparray (int) The number of particles counted so far.

`get_title`(*description=''*)
Get a string giving a title for the measurements.

Parameters

description: string, optional A description to be added to the title, if required.

add_comment: bool, optional Set to True if the comment string (if present) should be added to the title. The default is False.

Returns

title: string A title string

`integrate`(*flux*, *time*)
Integrate on a particle flux for a certain length of time. Calling this method adds one integration period.

Parameters

flux: array_like Photon flux array in particles per time unit. flux must either be None (indicating an integration on nothing) or be the same shape and size as the photon counter. The array must contain positive values - any negative values are treated as if they are zero.

time: float The integration time in time units. It must not be negative.

Raises

ValueError Raised if the photon flux is too large.

TypeError Raised if any of the parameters are of the wrong type, size or shape.

`plot` (*plotfig=None*, *plotaxis=None*, *datatype='image'*, *withbar=True*, *title=None*, *description=''*)
Plot the counter array. The subplot and show parameters allow more than one plot to be displayed within one plotting area (pyplot axis).

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot. If an axis is provided, the plot will be created within that axis. Providing an axis allows the caller to control exactly where a plot appears within a figure. If an axis is not provided, a new one will be created filling a new figure.

datatype: string, optional The type of data being displayed: 'matrix' or 'image'. 'matrix' is better for displaying an array (with square pixel boundaries) and 'image' is better for displaying a smoothly interpolated image. The default is 'image'.

withbar: bool, optional Set to False to suppress the colour bar. The default is True.

title: string, optional Optional title for the plot. The default is a string describing the PoissonIntegrator object. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

description: string, optional Additional description to be shown on the plot, if required.

Requires

miritools.miriplot matplotlib.pyplot

`poisson_noise_off`()
Disable Poisson noise. It will remain disabled until enabled with poisson_noise_on.

`poisson_noise_on`()
Enable Poisson noise. It will remain enabled until disabled with poisson_noise_off.

`readout` (*nsamples=1*)
Read the signal stored within the integrator and apply Poisson noise (if switched on).

Parameters

nsamples: int, optional, default=1 The number of samples from the Poisson distribution.

Returns

read_array: array_like The latest count as read out (with Poisson noise).

`reset` (*nresets=1*, *new_exposure=False*)
Reset the integrator. All internal counters are reset to zero (although persistence and zeropoint drift might leave a residual signal).

Parameters

nresets: int (optional) The number of resets to be applied. (This only makes a difference when there are persistence effects.) By default there will be one reset.

new_exposure: bool (optional) Set True if this reset begins a new exposure (and the integration count needs to be reset). By default this is False.

`set_seed` (*seedvalue=None*)
Set the seed for the numpy random number generator. This function can be used while testing to ensure the Poisson noise generated subsequently is well defined.

Parameters

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator. If not specified, a value of None will be sent, which randomises the seed.

wait(*time*)

Pause for a certain length of time without integrating.

Parameters

time: float The wait time in time units. It must not be negative.

Raises

ValueError Raised if the time is negative.

TypeError Raised if any of the parameters are of the wrong type, size or shape.

class miri.simulators.poisson_integrator.**ImperfectIntegrator**(*rows*, *columns*, *particle='photon'*, *time_unit='seconds'*, *bucket_size=None*, *simulate_poisson_noise=True*, *verbose=2*)

Class ImperfectIntegrator - Simulates the behaviour of a particle counting detector in which the actual number of particles (be they photons or electrons, etc...) detected varies with the expected number according to Poisson statistics.

The integrator is not perfect. Bad pixels, persistence, drift, latency and non-linear effects are simulated.

Parameters

rows: int The number of detector rows. Must be at least 1.

columns: int The number of detector columns. Must be at least 1.

particle: string, optional, default="photon" The name of the particles being counted.

time_unit: string, optional, default="seconds" The unit of time measurement.

bucket_size: int, optional, default=None The maximum number of particles that may be counted. If None there is no limit.

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

Attributes

persistence: float or tuple of floats A description of the persistence properties of the integrator.

- If a single float is provided, this is assumed to be a persistence factor in the range 0.0 to 1.0, where 0.0 means no persistence - signal is zeroed after reset. 1.0 means 100% persistence - reset has no effect on the signal.
- If a tuple of floats is provided this is taken as a set of polynomial coefficients describing a non-linear persistence effect (e.g. if the persistence is worse at high signals there may be a second order factor).

linearity: tuple of 2 floats The constant and slope of the function indicating how the sensitivity of the integrator varies at high counts compared to the well depth. [1.0, 0.0] means the integrator is perfectly linear [0.0, 0.0] means the integrator is completely dead [1.0, -1.0] means the counter starts at 100% sensitivity and reduces to 0% at full well. [1.0, -0.5] means the counter starts at 100% sensitivity and reduces to 50% at full well. [1.5, -1.0] means the counter starts at 100% sensitivity, remains level at 100% sensitivity until 50% full well, then reduces to 50% sensitivity at full well. The integrated sensitivity function gives the linearity function.

zp_slow: tuple of 2 floats or None: A list of coefficients describing the slow change in zeropoint as a function of clock time. For now, only a linear drift is simulated. The coefficients are [starting_dn, driftfactor], where starting_dn is a constant shift in DN and driftfactor is the amount of drift with time in DN/s. Set to None to turn off the long term drift altogether.

zp_fast: tuple of (tuple of floats) or None A 2-D array describing the short term zeropoint jumps as a function of integration number and flux. A tuple of floats is provided for each integration number, and each tuple of floats contains coefficients describing the zeropoint shift seen as a function of incident flux. For now, only a linear model is used, and each set of coefficients is described by 2 floats, [const, jumpfactor], where const is a ZP shift in DN which is always applied for that integration number and jumpfactor in s describes how the drift changes with source flux. Set to None to turn off zeropoint jumps altogether.

slow_latency_params: tuple of 2 floats or None The gain and decay factor of the slow latency function. Set to None to turn off the slow latency effect altogether.

fast_latency_parameters: tuple of 2 floats or None The gain and decay factor of the fast latency function. Set to None to turn off the fast latency effect altogether.

`hit_by_cosmic_ray`(*energy_map*, *row*, *column*)
Dump some cosmic ray energy onto a portion of the integrator.

Parameters

energy_map: array_like or int A single value or a 2-D array of cosmic ray energies in terms of equivalent particle count. The cosmic ray will affect the specified row and column plus neighbouring rows and columns, depending on the size of the energy map.

row: int The row on which the cosmic ray energy is centred.

column: int The column on which the cosmic ray energy is centred.

Raises

TypeError Raised if the energy map is invalid.

`leak`(*leakage*, *row*, *column*)
Simulate the leakage of some charge from the integrator, as might happen if there was a brief short circuit or malfunction.

Parameters

leakage: int The number of particles leaked from the current count.

row: int The row at which the leakage happens.

column: int The column at which the leakage happens.

`plot_linearity` (*plotfig=None*, *plotaxis=None*, *xmin=0*, *xmax=None*, *npoints=512*, *description=''*)
Plot integrator linearity as a function of count to a self-contained matplotlib figure.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot. If an axis is provided, the plot will be created within that axis. Providing an axis allows the caller to control exactly where a plot appears within a figure. If an axis is not provided, a new one will be created filling a new figure.

xmin: int, optional, default=0 The minimum number of counts to be plotted.

xmax: int, optional, default=bucket size or 100000 The maximum number of counts to be plotted.

npoints: int, optional, default=512 The number of points to be plotted.

description: string, optional Additional description to be shown on the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`plot_persistence` (*plotfig=None*, *plotaxis=None*, *xmin=0*, *xmax=None*, *npoints=512*, *description=''*)
Plot integrator persistence as a function of counts present when reset to a self-contained matplotlib figure.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot. If an axis is provided, the plot will be created within that axis. Providing an axis allows the caller to control exactly where a plot appears within a figure. If an axis is not provided, a new one will be created filling a new figure.

xmin: int, optional, default=0 The minimum number of counts to be plotted.

xmax: int, optional, default=bucket size or 100000 The maximum number of counts to be plotted.

npoints: int, optional, default=512 The number of points to be plotted.

description: string, optional Additional description to be shown on the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`plot_sensitivity` (*plotfig=None*, *plotaxis=None*, *xmin=0*, *xmax=None*, *npoints=512*, *description=''*)

Plot integrator sensitivity as a function of count to a self-contained matplotlib figure.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot. If an axis is provided, the plot will be created within that axis. Providing an axis allows the caller to control exactly where a plot appears within a figure. If an axis is not provided, a new one will be created filling a new figure.

xmin: int, optional, default=0 The minimum number of counts to be plotted.

xmax: int, optional, default=bucket size or 100000 The maximum number of counts to be plotted.

npoints: int, optional, default=512 The number of points to be plotted.

description: string, optional Additional description to be shown on the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`plot_zeropoint` (*fmin=0*, *fmax=500*, *nints=5*, *npoints=10*, *description=''*)

Plot integrator zero point drift as a function of flux and integration number to a self-contained matplotlib figure.

Parameters

fmin: int, optional, default=0 The minimum flux to be plotted.

fmax: int, optional, default=500 The maximum flux to be plotted.

nints: int, default=5 The number of integrations to be plotted.

npoints: int, optional, default=512 The number of points to be plotted.

description: string, optional Additional description to be shown on the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`set_bad_pixel_mask`(*mask*, *good_value=0*)

Sets the bad pixel mask associated with an imperfect integrator.

Parameters

mask: array_like A 2-D array of containing a bad pixel mask where good pixels are indicated by zero values and bad pixels by non-zero values. The mask must have the same size and shape as the integrator. If mask=None, the bad pixel mask is cleared.

good_value: int, optional, default=0 The value used to represent a good pixel. Anything else is taken as bad.

Raises

TypeError Raised if the bad pixel mask is invalid.

`set_latency`(*slow_parameters*, *fast_parameters*)

Defines integrator latency effects. Latency effects are caused by the integrator retaining a memory of previous integrations and taking a finite time to react to changes. There are several latency effects covering different timescales:

- Slow latency effects are caused by a build up of sensitivity during exposures, leading to each successive integration having a slightly greater slope than the previous integration. The magnitude of the effect depends on the source brightness and time for which each source is observed. The gain factor is given in (1/counts). The decay factor is given in seconds.
- Fast latency effects are caused by rapid changes in flux. Instead of the flux changing instantly from one level to another, the integrator behaves as if the flux level is decaying exponentially between one level and the next. The gain factor is dimensionless. The decay factor is given in seconds.

Both latency effects are described by two parameters:

- The gain factor describes the magnitude of the effect.
- The decay factor describes how quickly the effect decays.

Parameters

slow_parameters: tuple of 2 floats or None The gain and decay factor of the slow latency function. Set to None to turn off the slow latency effect altogether.

fast_parameters: tuple of 2 floats or None The gain and decay factor of the fast latency function. Set to None to turn off the fast latency effect altogether.

Raises

TypeError or AssertionError Raised if either of the latency parameters are invalid.

`set_linearity`(*linearity*)

Define the integrator linearity effects.

Parameters

linearity: tuple of 2 floats or None The constant and slope of the function indicating how the sensitivity of the integrator varies at high counts compared to the well depth. None means the integrator is perfectly linear [1.0, 0.0] means the integrator is perfectly linear [0.0, 0.0] means the integrator is completely dead [1.0, -1.0] means the counter starts at 100% sensitivity and reduces to 0% at full well. [1.0, -0.5] means the counter starts at 100% sensitivity and reduces to 50% at full well. [1.5, -1.0] means the counter starts at 100% sensitivity, remains level at 100% sensitivity until 50% full well, then reduces to 50% sensitivity at full well. The integrated sensitivity function gives the linearity function.

`set_persistence`(*persistence*)

Define the integrator persistence effects. Persistence effects are observed if a reset() does not completely clear the count from a previous integration.

Parameters

persistence: float or tuple of floats or None A description of the persistence properties of the integrator.

- If a single float is provided, this is assumed to be a persistence factor in the range 0.0 to 1.0, where 0.0 means no persistence - signal is zero after reset. 1.0 means 100% persistence - reset has no effect on the signal.
- If a tuple of floats is provided this is taken as a set of polynomial coefficients describing a non-linear persistence effect (e.g. if the persistence is worse at high signals there may be a second order factor).
- Set to None to turn off persistence effects altogether (equivalent to setting it to 0.0).

Raises

TypeError Raised if the persistence parameter is invalid.

`set_zeropoint`(*zp_slow*, *zp_fast*)

Define the integrator zeropoint effects. Zeropoint effects are observed if the counter does not start counting from zero after a reset. The zeropoint can vary with time, source brightness and integration number.

Parameters

zp_slow: tuple of 2 floats or None: A list of coefficients describing the slow change in zeropoint as a function of clock time. For now, only a linear drift is simulated. The coefficients are [starting_dn, driftfactor], where starting_dn is a constant shift in DN and driftfactor is the amount of drift with time in DN/s. Set to None to turn off the long term drift altogether.

zp_fast: tuple of (tuple of floats) or None A 2-D array describing the short term zeropoint jumps as a function of integration number and flux. A tuple of floats is provided for each integration number, and each tuple of floats contains coefficients describing the zeropoint shift seen as a function of incident flux. For now, only a linear model is used, and each set of coefficients is described by 2 floats, [const, jumpfactor], where const is a ZP shift in DN which is always applied for that integration number and jumpfactor in s describes how the drift changes with source flux. Set to None to turn off zeropoint jumps altogether.

Raises

TypeError or AssertionError Raised if the zeropoint coefficients parameter is invalid.

2.2.3 Functions

None.

CHAPTER THREE

SUBPACKAGE - MIRIMSIM

The MIRI imager simulator.

3.1 MIRI Imager Simulator (`miri.simulators.mirimsim`)

Release 0.5devxx

Date June 26, 2014

3.1.1 LICENCE

Copyright

Terms and Conditions of Use

3.1.2 Introduction

MIRI imager simulator.

DOCUMENTATION TO BE INSERTED HERE

References

3.1.3 Scripts

The mirimsim scripts may be found in the mirimsim/scripts directory.

mirimsim

3.1.4 Modules

The mirimsim modules may be found in the mirimsim/lib directory.

CHAPTER FOUR

SUBPACKAGE - SCASIM

The MIRI SCA simulator.

4.1 MIRI Sensor Chip Assembly Simulator (`miri.simulators.scasim`)

Release 0.5devxx

Date June 26, 2014

4.1.1 LICENCE

Copyright

Terms and Conditions of Use

Disclaimer

4.1.2 Introduction

This document describes the implementation details for the MIRI Sensor Chip Assembly (SCA) simulator software. It should be read alongside the "MIRI Sensor Chip Assembly (SCA) Simulator User Manual" (1), which describes how to use the software, and the "MIRI Sensor Chip Assembly (SCA) Simulator Software Design Document" (2), which describes the design on which this software implementation is based.

The SCA simulator software is designed to be run from the scripts - e.g. see the description of the scasim script in "Scripts" section below. The scripts are given basic information, such as the input and output file names, the detector temperature and cosmic ray environment to be simulated and the detector readout and subarray modes required. It is also possible to call the SCA simulator directly from Python software - see the description of the simulate_sca_fromdata function within the "sensor_chip_assembly" module. More detailed information about the properties of the SCA (such as how the dark current and read noise changes with temperature) is contained within configuration files - see the "Data" section for details. The "Modules" section describes the detailed contents of the software modules.

The software depends on the tools and utilities in the miritools module.

References

1. MIRI Sensor Chip Assembly (SCA) Simulator User Manual, V0.98, 23 November 2011.
2. MIRI Sensor Chip Assembly (SCA) Simulator Software Design Document, V0.91, 27 September 2011.
3. JPL D-25632, MIRI Operations Concept Document (Revision C), Christine Chen, Karl Gordon, Scott Friedman and Margaret Meixner, STScI, 4 March 2010.
4. JPL D-46944, MIRI Flight Focal Plane Module End Item Data Package (FPM EIDP) (edited version), A. Schneider et al., Initial X1, 10 May 2010
5. The JWST MIRI Instrument Concept, Wright et al., Proc. SPIE, 5487, 653, 2004.
6. MIRI DFM 308-04.02, Analysis of the Proton Flux on the MIRI Detector Arrays, M. Ressler, 3 August 2009.
7. JWST-STScI-00198, SM-12, A Library of Simulated Cosmic Ray Events Impacting JWST HgCdTe Detectors, M. Robberto, 9 March 2010. (MIRI update in press)
8. JWST Calibration Software Online Documentation, http://stsdas.stsci.edu/jwst/docs/sphinx/
9. DHAS miri_sloper Online Documentation, http://tiamat.as.arizona.edu/dhas/
10. FM Raw Science Data Definition (Issue 1, Draft B), Tim Grundy, 12 January 2011.
11. MIRI calibration file metadata convention, Jane Morrison, 9 June 2011.

12. MIRI Test Report, Description of Bad Pixel Mask, Jane Morrison, V4, 28 August 2011

4.1.3 Scripts

The scasim scripts may be found in the scasim/scripts directory.

scasim

make_sca_file

make_bad_pixel_mask

make_maps_from_dhas

make_sca_calibration

4.1.4 Configuration Information

The parameters provided to the scripts, described in the previous section, are used to control those aspects of a MIRI simulation that are expected to change from observation to observation. More detailed control of the simulations is possible by editing the configuration files supplied with the simulator. The files, which may be found in the scasim/lib directory, are:

```
cosmic_ray_properties.py
detector_properties.py
amplifier_properties.py
```

These files respectively describe the cosmic ray environment, the properties of the detector and the properties of the readout amplifiers and electronics.

To make a change to one or more of these files, copy them to your current working directory and edit them. As long as the new files are within your current working directory (or a subdirectory within it), there is no need to rebuild the SCAsim software to make the new parameters current. To revert to the original parameters, simple delete or rename the edited files.

SCASim will search the current directory, the MIRI data directories and then the PYTHONPATH directories for these files, and it will report the full path and name of each one used. Make sure SCAsim is using the files you expect.

NOTE: These configuration files are processed by the Python interpreter as if they were source code. Be careful not to miss out matching quotes or brackets. If there is a problem in interpreting one of these files, you will see a parsing error when SCAsim is executed. The offending file will be named. See the “MIRI Sensor Chip Assembly (SCA) Simulator User Manual” for details.

4.1.5 Shared Modules

The following modules may be found in the top level simulators package, and are shared with other MIRI simulators.

4.1.6 Modules

The scasim modules may be found in the scasim/lib directory.

Sensor chip assembly module (`miri.simulators.scasim.sensor_chip_assembly`)

Description

The sensor_chip_assembly module contains the SensorChipAssembly class (the top level class of the MIRI Sensor Chip Assembly simulator) plus associated functions. The top level simulate_sca function runs the SCA simulator; reading a detector illumination input file and writing an output file. The simulate_sca_fromdata function provides a way of running the SCA simulator directly from data, without the need to write a detector illumination file. The overall class hierarchy is:

```
SensorChipAssembly
    CosmicRayEnvironment
        CosmicRay
    IlluminationMap
        DataMap
    BadPixelMap
        DataMap
    DarkMap
        DataMap
    ExposureData
    DetectorArray
        PoissonIntegrator
        Amplifier
        MeasuredVariable

    ParameterFileManager
    QuantumEfficiency
        Filter
```

The MeasuredVariable, QuantumEfficiency and ParameterFileManager classes are obtained from the miritools package. The software also uses the following configuration files to describe the properties of the MIRI Sensor Chip Assembly and its environment:

```
cosmic_ray_properties.py
detector_properties.py
amplifier_properties.py
```

Objects

class `miri.simulators.scasim.sensor_chip_assembly.`**`SensorChipAssembly`**(*detectorid*, *readout_mode='FAST'*, *subarray='FULL'*, *inttime=None*, *ngroups=None*, *nints=None*, *temperature=6.7*, *cosmic_ray_mode='SOLAR_MIN'*, *fileformat='STSCI'*, *qe_adjust=True*, *simulate_poisson_noise=True*, *simulate_read_noise=True*, *simulate_ref_pixels=True*, *simulate_badpixels=True*, *simulate_dark_current=True*, *simulate_amp_effects=True*, *version=''*, *makeplot=False*, *verbose=2*)

Class Sensor Chip Assembly - Simulates the behaviour of the MIRI detectors. The class converts detector illumination data (typically provided in a FITS file generated by another MIRI simulator and read by the read_data method) and generates simulated detector data, which may be written to a FITSWriter or level 1 FITS file by the write_data method.

Parameters

detectorid: string The detector ID, identifying a particular detector. The MIRI instrument has three detectors: 'IM', 'LW' and 'SW'. (These correspond to the imager and the long and short wavelength arms of the MRS respectively.)

readout_mode: string, optional, default='FAST' Readout mode, which can be one of the following common options:

- 'SLOW' - 10 samples per readout and defaults of ngroups=10 and nints=1
- 'FAST' - 1 sample per readout and defaults of ngroups=1 and nints=10
- 'FASTINTAVG' - same as FAST but with groups of 4 integrations averaged to reduce data volume.
- 'FASTGRPAVG' - same as FAST but with groups of 4 groups averaged to reduce data volume.

The following unusual options are also available for testing:

- 'FASTGRPGAP' - a non-MIRI mode similar to FAST mode but with 4 frames per group and a gap of 8 frames between each group.
- 'SLOWINTAVG' - same as SLOW but with groups of 4 integrations averaged and default ngroups=1.
- 'SLOWGRPAVG' - same as SLOW but with groups of 4 groups averaged and default ngroups=4.
- 'SLOWGRPGAP' - a non-MIRI mode similar to SLOW mode but with 4 frames per group and a gap of 8 frames between each group.

subarray: string, optional, default='FULL' Detector subarray mode for output. This can be one 'FULL', 'MASK1550', 'MASK1140', 'MASK1065', 'MASKLYOT', 'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64' or 'SLITNESSPRISM', etc. 'FULL' is full-frame and the other modes read out portions of the detector as described in the MIRI Operational Concept Document. The simulator can also accept the other subarray modes defined in detector_properties.py for test purposes.

inttime: float, optional The integration time in seconds to be simulated. This parameter will be ignored if ngroups is provided as well. *NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.*

ngroups: int, optional The number of readout groups. If None, this is derived from the integration time and readout mode. Otherwise the parameter overrides the integration time and sets the number of groups directly. There must be at least one group.

nints: int, optional The number of integrations per exposure. It must be at least 1. If None, the default set by the readout mode is used. The total exposure time is nints x inttime.

temperature: float, optional, default=6.7 The temperature of the detector, which will determine the dark current and read noise.

cosmic_ray_mode: string, optional, default='SOLAR_MIN' The cosmic ray environment mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

qe_adjust: boolean, optional, default=True A flag that may be used to switch off quantum efficiency adjustment (for example to observe what effects in a simulation are caused by QE).

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_ref_pixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of reference pixels in the data.

simulate_badpixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of bad pixels in the data, even if a bad pixel map containing bad pixels is specified in the detector properties.

simulate_dark_current: boolean, optional, default=True A flag that may be used to switch off the addition of dark current (for example to observe what effects in a simulation are caused by dark current).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the ratio of DNs to electrons is exactly 1.0*

version: string, optional Software version string.

makeplot: boolean, optional, default=False Plotting flag. Activates plotting of data when True.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

KeyError Raised if any of the initialisation parameters do not contain a recognised keyword.

`amp_effects_off``()`
Disable amplifier bias, gain and non-linearity. They will remain disabled until enabled with amp_effects_on.

`amp_effects_on``()`
Enable amplifier bias, gain and non-linearity. They will remain enabled until disabled with amp_effects_off.

`bad_pixels_off``()`
Disable bad pixels. They will remain disabled until enabled with bad_pixels_on.

`bad_pixels_on``()`
Enable bad pixels. They will remain enabled until disabled with bad_pixels_off.

`cosmic_ray_str`(*prefix*=''')
Returns a string describing the cosmic ray environment.

`dark_current_off``()`
Disable dark current. It will remain disabled until enabled with dark_current_on.

`dark_current_on``()`
Enable dark current. It will remain enabled until disabled with dark_current_off.

`define_cosmic_ray_env`(*cosmic_ray_mode*)
Define the cosmic ray environment for the simulation.

Parameters

cosmic_ray_mode: string, optional, default='SOLAR_MIN' The cosmic ray environment mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

exposure(*nints=None*, *frame_time=None*)
Simulate a detector exposure consisting of one or more integrations, each of which consists of one of more groups of readouts.

Parameters

nints: int, optional The number of integrations in this exposure. It must be at least 1. If None, the default set by the readout mode is used. The total exposure time is nints x integration time.

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the frame time obtained from the current detector readout mode and subarray. This can be used, for example, to simulate the timing of full frame exposures with subarray-sized data (to save memory). If None, the frame time obtained from the current readout mode and subarray is used.

Returns

integration_data: array_like uint32 An array of data from the final integration.

Prerequisites

Can only be used after the detector illumination data have been imported by the read_data method. The detector readout mode and integration time should have been defined with the set_readout_mode method.

Raises

AttributeError Raised if the read_data method has not been executed first.

ValueError Raised if any of the parameters are out of range.

integration(*intnum*, *frame_time=None*)
Simulate a detector integration consisting of one or more groups of readouts.

Parameters

intnum: int Integration number, which determines where data are stored in the simulated science data structure. Must be 0 or greater.

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the frame time obtained from the current detector readout mode and subarray. This can be used, for example, to simulate the timing of full frame exposures with subarray-sized data (to save memory). If None, the frame time obtained from the current readout mode and subarray is used.

Returns

integration_data: array_like uint32 An array of integration data.

Prerequisites

Can only be used after the detector illumination data have been imported by the read_data method. The detector readout mode and integration time should have been defined with the set_readout_mode method.

Raises

AttributeError Raised if the read_data method has not been executed first.

ValueError Raised if any of the parameters are out of range.

plot(*description=''*)
Plot the simulated science data. Each set of integration data is plotted in a separate pyplot figure and the image read out from each group is plotted in a separate subplot.

Parameters

description: string, optional Additional description to be shown on the plot, if required.

Requires

matplotlib.pyplot

ImportError Raised if the matplotlib plotting library is not available.

plot_ramp(*row*, *column*, *description=''*, *frame_time=None*, *show_ints=False*)
Plot a ramp of signal vs integration and group at a particular row,column of the simulated data.

Parameters

row: int The row at which the ramp is to be plotted.

column: int The column at which the ramp is to be plotted.

description: string, optional Additional description to be shown on the plot, if required.

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter allows the frame time obtained from the detector properties to be overridden. This can be used, for example, to simulate the timing of full frame data with a subarray. If None, the frame time obtained from the detector properties for the current subarray and readout mode is used.

show_ints: boolean, optional, default=False Set to True to distinguish the ramps belonging to different integrations. Only works when a single row and column is specified.

poisson_noise_off()
Disable Poisson noise. It will remain disabled until enabled with poisson_noise_on.

poisson_noise_on()
Enable Poisson noise. It will remain enabled until disabled with poisson_noise_off.

read_data(*filename*, *ftype='STSCI'*, *scale=1.0*)
Read the detector illumination data from a file.

Parameters

filename: string The name of the file to be opened.

ftype: string, optional, default='STSCI' The type of file from which to read the data ('STSCI', 'FITS', 'ASCII' or 'IMAGE'). Only 'STSCI' or 'FITS' can provide a header and sensible wavelength information. Passed to the data_maps module.

scale: float, optional, default=1.0 An optional scale factor to apply to the intensity data. This is for debugging and testing only - the input data should already be scaled to photons/second/pixel.

Raises

ValueError Raised if any of the parameters are out of range.

TypeError Raised if any of the parameters are of the wrong type, size or shape.

See also

The documentation for the IlluminationData class of the data_maps package.

`read_fringe_map` (*filename*, *ftype='STSCI'*)
Read fringe map data from a file.

Parameters

filename: string The name of the file to be opened.

ftype: string, optional, default='STSCI' The type of file from which to read the data ('STSCI', 'FITS', 'ASCII' or 'IMAGE'). Passed to the data_maps module.

See also

The documentation for the DataMap class of the data_maps package.

Prerequisites

Can only be used after the detector illumination data have been imported by the read_data method.

Raises

AttributeError Raised if the read_data method has not been executed first.

`read_noise_off` ()
Disable read noise. It will remain disabled until enabled with read_noise_on.

`read_noise_on` ()
Enable read noise. It will remain enabled until disabled with read_noise_off.

`readout_mode_str` (*prefix=''*)
Returns a string describing the readout mode.

`set_data` (*intensity*, *scale=None*, *wavelength=None*, *metadata=None*, *intensity_metadata=None*, *wavelength_metadata=None*, *subarray=None*)
Define the detector illumination data arrays directly, as an alternative to reading a file. This function is useful when calling scasim functions directly from other Python software.

Parameters

intensity: array_like An array describing the illumination reaching the detector pixels (in photons per second). The array can be 2 dimensional of shape (rows, columns) or 3 dimensional of shape (slices, rows, columns). In both cases the number of rows and columns indicates the detector size in pixels. The third dimension is optional and can be used to divide the illumination into slices at different wavelengths.

scale: float, optional, default=None An optional scale factor to apply to the intensity data. If not provided, no scaling is applied. This is for debugging and testing only - the input data should already be scaled to photons/second/pixel.

wavelength: array_like, optional An array describing the wavelength of the illumination reaching the detector pixels (in microns). If None, this array is ignored; otherwise the wavelength array must match the intensity array in one of 3 ways:

1. The shapes must match exactly; or
2. **The wavelength array is 2 dimensional of shape (rows,** columns) which exactly match the number of rows and columns in the intensity array; or
3. **The wavelength array is 3 dimensional of shape** (slices, rows, columns) where the number of rows and columns is 1 and the number of slices matches the intensity array.

metadata: pyfits metadata object, optional The primary FITS metadata describing the data. If None, the metadata is ignored.

intensity_metadata: pyfits metadata object, optional FITS metadata contained in the INTENSITY extension. If None, the metadata is ignored.

wavelength_metadata: pyfits metadata object, optional FITS metadata contained in the WAVELENGTH extension. If None, the metadata is ignored.

subarray: string, optional The subarray mode of the input data. If not specified, the subarray mode will be taken from the supplied metadata, and if neither are present it will be assumed full frame.

`set_readout_mode` (*mode*, *inttime=None*, *ngroups=None*, *nints=None*, *nsample=None*)

Defines the SCA detector readout mode and integration parameters.

Parameters

mode: string Readout mode, which can be one of the following options.

- 'SLOW' - 10 samples per readout and defaults of ngroups=10 and nints=1
- 'FAST' - 1 sample per readout and defaults of ngroups=1 and nints=10
- 'FASTINTAVG' - same as FAST but with groups of 4 integrations averaged to reduce data volume.
- 'FASTGRPAVG' - same as FAST but with groups of 4 groups averaged to reduce data volume.

inttime: float, optional The integration time in seconds. This parameter also defines the number of readout groups, unless ngroups is specified. The integration time must be positive, as there must be at least one group. *NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.*

ngroups: int, optional The number of readout groups. If None, this is derived from the integration time and readout mode. If specified, this parameter defines the number of groups and integration time and overrides the inttime parameter. There must be at least one group.

nints: int, optional The number of integrations required. If None, the default for the readout mode will be used. There must be at least one integration. *NOTE: nints can be safely changed without affecting the readout mode.*

nsample: int, optional Normally None, but if specified allows nsample to be defined directly - overriding the values defined in the detector properties file. This is only for testing weird readout modes, since MIRI users cannot change nsample explicitly. There must be at least one sample. *NOTE: Altering this value from its default will change the readout mode to 'TESTnn'.*

Raises

ValueError Raised if any of the parameters are out of range.

`set_seed` (*seedvalue=None*)

Set the seed for the numpy random number generator. This function can be used while testing to ensure the randomised effects that follow are well defined.

Parameters

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator. If not specified, a value of None will be sent, which randomises the seed.

temperature_str(*prefix=''*)
Returns a string describing the detector temperature

write_data(*filename*, *datashape='cube'*, *clobber=False*)
Write the simulated data to a given output FITS file.

Parameters

filename: string The name of the file to be created.

datashape: string, optional, default='cube' The SCI data format to be written (not valid for STSCI format).

- 'hypercube' - write the SCI data to a 4 dimensional FITS image with separate columns x rows x groups x integrations dimensions.
- 'cube' - append the groups and integrations to make a 3 dimensional FITS image with columns x rows x (groups and integrations) dimensions.

clobber: bool, optional, default=False Parameter passed to pyfits.HDUlist.writeto

Prerequisites

Can only be used after the exposure data have been generated by the integration or exposure methods.

Raises

AttributeError Raised if no exposure data exists.

Functions

`miri.simulators.scasim.sensor_chip_assembly.`**`simulate_sca`**`(`*inputfile*, *outputfile*, *detectorid*, *scale=1.0*, *fringemap=None*, *readout_mode=None*, *subarray=None*, *frame_time=None*, *inttime=None*, *ngroups=None*, *nints=None*, *temperature=None*, *cosmic_ray_mode=None*, *fileformat='STSCI'*, *datashape='cube'*, *clobber=False*, *qe_adjust=True*, *simulate_poisson_noise=True*, *simulate_ref_pixels=True*, *simulate_read_noise=True*, *simulate_badpixels=True*, *simulate_dark_current=True*, *simulate_amp_effects=True*, *version=''*, *makeplot=False*, *seedvalue=None*, *verbose=2*`)`

Runs the MIRI SCA simulator on the given input file, generating the given output file.

Parameters

inputfile: string The name of the FITS file containing detector illumination data (normally created by another MIRI simulator).

outputfile: string The name of the FITSWriter or level 1 FITS file to contain the simulated SCA data.

detectorid: string Detector ID, identifying a particular detector. The MIRI instrument has detectors: 'IM', 'LW' and 'SW'. (These correspond to the imager and the long and short wavelength arms of the MRS respectively.)

scale: float, optional, default=1.0 An optional scale factor to apply to the intensity data. This is for debugging and testing only - the input data should already be scaled to photons/second/pixel.

fringemap: string, optional The name of a file containing a fringe map to be used to modulate the input illumination. If not specified, no fringe map will be used.

readout_mode: string, optional Readout mode, which can be one of the following common options:

- 'SLOW' - 10 samples per readout and defaults of ngroups=10 and nints=1.
- 'FAST' - 1 sample per readout and defaults of ngroups=1 and nints=10.
- 'FASTINTAVG' - same as FAST but with groups of 4 integrations averaged.

- 'FASTGRPAVG' - same as FAST but with groups of 4 groups averaged.

The following unusual options are also available for testing:

- 'FASTGRPGAP' - a non-MIRI mode similar to FAST mode but with 4 frames per group and a gap of 8 frames between each group.
- 'SLOWINTAVG' - same as SLOW but with groups of 4 integrations averaged and default ngroups=1.
- 'SLOWGRPAVG' - same as SLOW but with groups of 4 groups averaged and default ngroups=4.
- 'SLOWGRPGAP' - a non-MIRI mode similar to SLOW mode but with 4 frames per group and a gap of 8 frames between each group.

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the detector properties default.

subarray: string, optional Detector subarray mode for output. This can be one 'FULL', 'MASK1550', 'MASK1140', 'MASK1065', 'MASKLYOT', 'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64' or 'SLITNESSPRISM', etc. 'FULL' is full-frame and the other modes read out portions of the detector as described in the MIRI Operational Concept Document. The simulator can also accept the other subarray modes defined in detector_properties.py for test purposes. If this parameter is not explicitly given it will be obtained from the FITS header of the input file (unless the input data specifies a non-standard subarray). Failing that, it will be obtained from the detector properties default (FULL).

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the frame time obtained from the detector readout mode and subarray. This can be used, for example, to simulate the timing of full frame exposures with subarray-sized data (to save memory). If None, the frame time obtained from the current readout mode and subarray is used.

inttime: float, optional The integration time in seconds to be simulated. This parameter will be ignored if ngroups is provided as well. *NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.*

ngroups: int, optional The number of readout groups. If None, this is derived from the integration time and readout mode. Otherwise the parameter overrides the integration time and sets the number of groups directly. There must be at least one group.

nints: int, optional The number of integrations per exposure. It must be at least 1. If None, the default set by the readout mode is used. The total exposure time is nints x inttime.

temperature: float, optional, default=detector target temperature The temperature of the detector, which will determine the dark current and readout noise.

cosmic_ray_mode: string, optional, default=cosmic ray properties The cosmic ray environment mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the cosmic ray properties default.

fileformat: string, optional, default='STSCI' The kind of file format to be written.

- 'STSCI' - use the NEW STScI level 1 model for the JWST DMS pipeline.
- 'FITSWriter' - emulate the format written by the FITSWriter during MIRI VM and FM tests and read by the DHAS.

- 'level1' - emulate the level 1 FITS format of the OLD MIRI data model (backwards compatibility only).

datashape: string, optional, default='cube' The SCI data format to be written.

- 'hypercube' - write the SCI data to a 4 dimensional FITS image with separate columns x rows x groups x integrations dimensions.
- 'cube' - append the groups and integrations to make a 3 dimensional FITS image with columns x rows x (groups and integrations) dimensions.

clobber: bool, optional, default=False Parameter passed to pyfits.HDUlist.writeto

qe_adjust: boolean, optional, default=True A flag that may be used to switch off quantum efficiency adjustment (for example to observe what effects in a simulation are caused by QE).

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

simulate_ref_pixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of reference pixels in the data.

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_badpixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of bad pixels in the data, even if a bad pixel map containing bad pixels is specified in the detector properties.

simulate_dark_current: boolean, optional, default=True A flag that may be used to switch off the addition of dark current (for example to observe what effects in a simulation are caused by dark current).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the output data is in electrons rather than DNs.*

version: string, optional Software version string.

makeplot: boolean, optional, default=False Plotting flag. Activates plotting of data when True.

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator before generating the test data. If not specified, a value of None will be sent, which randomises the seed.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the simulation parameters are out of range. Also raised if the value of a parameter is invalid or inappropriate.

TypeError Raised if any of the simulation parameters are of the wrong type, size or shape.

KeyError Raised if any of the simulation parameters do not contain a recognised keyword.

IndexError Raised if an attempt is made to access beyond the bounds of the data.

IOError Raised if a file cannot be read, interpreted or written.

ImportError A delayed ImportError is raised if there is an attempt to use an optional library (such as matplotlib) which is not available. The software fails immediately if a compulsory library is not available.

AttributeError Raised if simulation methods are executed in the wrong order, meaning that a necessary attribute is missing. A programming error.

`miri.simulators.scasim.sensor_chip_assembly.`**`simulate_sca_list`**`(`*inputfile*, *outputfile*, *detectorid*, *scale=1.0*, *fringemap=None*, *readout_mode=None*, *subarray=None*, *frame_time=None*, *inttime=None*, *ngroups=None*, *nints=None*, *temperature=None*, *cosmic_ray_mode=None*, *fileformat='STSCI'*, *datashape='cube'*, *clobber=False*, *qe_adjust=True*, *simulate_poisson_noise=True*, *simulate_ref_pixels=True*, *simulate_read_noise=True*, *simulate_badpixels=True*, *simulate_dark_current=True*, *simulate_amp_effects=True*, *version=''*, *makeplot=False*, *seedvalue=None*, *verbose=2*`)`

Runs the MIRI SCA simulator on the given list of input files, generating the given list of output files. Persistence effects are simulated between one exposure and the next.

Parameters

inputfile: string or list of strings The name of the FITS file containing detector illumination data (normally created by another MIRI simulator).

outputfile: string or list of strings The name of the FITSWriter or level 1 FITS file to contain the simulated SCA data. If an entry in the list is empty, no file will be written.

detectorid: string The detector ID, identifying a particular detector. The MIRI instrument has three sensor chip assemblies: 'IM', 'LW' and 'SW'. (These correspond to the imager and the long and short wavelength arms of the MRS respectively.)

scale: float, optional, default=1.0 An optional scale factor to apply to the intensity data. This is for debugging and testing only - the input data should already be scaled to photons/second/pixel.

fringemap: string, optional The name of a file containing a fringe map to be used to modulate the input illumination. If not specified, no fringe map will be used.

readout_mode: string, optional Readout mode, which can be one of the following common options:

- 'SLOW' - 10 samples per readout and defaults of ngroups=10 and nints=1.
- 'FAST' - 1 sample per readout and defaults of ngroups=1 and nints=10.
- 'FASTINTAVG' - same as FAST but with groups of 4 integrations averaged.
- 'FASTGRPAVG' - same as FAST but with groups of 4 groups averaged.

The following unusual options are also available for testing:

- 'FASTGRPGAP' - a non-MIRI mode similar to FAST mode but with 4 frames per group and a gap of 8 frames between each group.
- 'SLOWINTAVG' - same as SLOW but with groups of 4 integrations averaged and default ngroups=1.
- 'SLOWGRPAVG' - same as SLOW but with groups of 4 groups averaged and default ngroups=4.
- 'SLOWGRPGAP' - a non-MIRI mode similar to SLOW mode but with 4 frames per group and a gap of 8 frames between each group.

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the detector properties default.

subarray: string, optional Detector subarray mode for output. This can be one 'FULL', 'MASK1550', 'MASK1140', 'MASK1065', 'MASKLYOT', 'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64' or 'SLITNESSPRISM', etc. 'FULL' is full-frame and the other modes read out portions of the detector as described in the MIRI Operational Concept Document. The simulator can also accept the other subarray modes defined in detector_properties.py for test purposes. If this parameter is not explicitly given it will be obtained from the FITS header of the input file (unless the input data specifies a non-standard subarray). Failing that, it will be obtained from the detector properties default (FULL).

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the frame time obtained from the detector readout mode and subarray. This can be used, for example, to simulate the timing of full frame exposures with subarray-sized data (to save memory). If None, the frame time obtained from the current readout mode and subarray is used.

inttime: float, optional The integration time in seconds to be simulated. This parameter will be ignored if ngroups is provided as well. *NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.*

ngroups: int, optional The number of readout groups. If None, this is derived from the integration time and readout mode. Otherwise the parameter overrides the integration time and sets the number of groups directly. There must be at least one group.

nints: int, optional The number of integrations per exposure. It must be at least 1. If None, the default set by the readout mode is used. The total exposure time is nints x inttime.

temperature: float, optional, default=detector target temperature The temperature of the detector, which will determine the dark current and readout noise.

cosmic_ray_mode: string, optional, default=cosmic ray properties The cosmic ray environment mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum

- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the cosmic ray properties default.

fileformat: string, optional, default='STSCI' The kind of file format to be written.

- 'STSCI' - use the NEW STScI level 1 model for the JWST DMS pipeline.
- 'FITSWriter' - emulate the format written by the FITSWriter during MIRI VM and FM tests and read by the DHAS.
- 'level1' - emulate the level 1 FITS format of the OLD MIRI data model (backwards compatibility only).

datashape: string, optional, default='cube' The SCI data format to be written.

- 'hypercube' - write the SCI data to a 4 dimensional FITS image with separate columns x rows x groups x integrations dimensions.
- 'cube' - append the groups and integrations to make a 3 dimensional FITS image with columns x rows x (groups and integrations) dimensions.

clobber: bool, optional, default=False Parameter passed to pyfits.HDUlist.writeto

qe_adjust: boolean, optional, default=True A flag that may be used to switch off quantum efficiency adjustment (for example to observe what effects in a simulation are caused by QE).

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

simulate_ref_pixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of reference pixels in the data.

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_badpixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of bad pixels in the data, even if a bad pixel map containing bad pixels is specified in the detector properties.

simulate_dark_current: boolean, optional, default=True A flag that may be used to switch off the addition of dark current (for example to observe what effects in a simulation are caused by dark current).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the output data is in electrons rather than DNs.*

version: string, optional Software version string.

makeplot: boolean, optional, default=False Plotting flag. Activates plotting of data when True.

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator before generating the test data. If not specified, a value of None will be sent, which randomises the seed.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary

- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the simulation parameters are out of range. Also raised if the value of a parameter is invalid or inappropriate.

TypeError Raised if any of the simulation parameters are of the wrong type, size or shape.

KeyError Raised if any of the simulation parameters do not contain a recognised keyword.

IndexError Raised if an attempt is made to access beyond the bounds of the data.

IOError Raised if a file cannot be read, interpreted or written.

ImportError A delayed ImportError is raised if there is an attempt to use an optional library (such as matplotlib) which is not available. The software fails immediately if a compulsory library is not available.

AttributeError Raised if simulation methods are executed in the wrong order, meaning that a necessary attribute is missing. A programming error.

`miri.simulators.scasim.sensor_chip_assembly.`**`simulate_sca_fromdata`**`(`*intensity*, *outputfile*, *detectorid*, *scale=1.0*, *wavelength=None*, *metadata=None*, *intensity_metadata=None*, *wavelength_metadata=None*, *fringemap=None*, *readout_mode=None*, *subarray=None*, *frame_time=None*, *inttime=None*, *ngroups=None*, *nints=None*, *temperature=None*, *cosmic_ray_mode=None*, *fileformat='STSCI'*, *datashape='cube'*, *clobber=False*, *qe_adjust=True*, *simulate_poisson_noise=True*, *simulate_ref_pixels=True*, *simulate_read_noise=True*, *simulate_badpixels=True*, *simulate_dark_current=True*, *simulate_amp_effects=True*, *version=''*, *makeplot=False*, *seedvalue=None*, *verbose=2*`)`

Runs the MIRI SCA simulator on the given input data, generating the given output file. This is an alternative to simulate_sca, which can be used when the detector illumination data arrays are already available in memory.

Parameters

intensity: array_like An array describing the illumination reaching the detector pixels (in photons per second). The array can be 2 dimensional of shape (rows, columns) or 3 dimensional of shape (slices, rows, columns). In both cases the number of rows and columns indicates the detector size in pixels. The third dimension is optional and can be used to divide the illumination into slices at different wavelengths.

outputfile: string The name of the FITSWriter or level 1 FITS file to contain the simulated SCA data.

detectorid: string or int The detector ID, identifying a particular detector. The MIRI instrument has three sensor chip assemblies: 'IM', 'LW' and 'SW'. (These correspond to the imager and the long and short wavelength arms of the MRS respectively.)

scale: float, optional, default=1.0 An optional scale factor to apply to the intensity data. This is for debugging and testing only - the input data should already be scaled to photons/second/pixel.

wavelength: array_like, optional An array describing the wavelength of the illumination reaching the detector pixels (in microns). If None, this array is ignored; otherwise the wavelength array must match the intensity array in one of 3 ways:

1. The shapes must match exactly; or
2. **The wavelength array is 2 dimensional of shape (rows,** columns) which exactly match the number of rows and columns in the intensity array; or
3. **The wavelength array is 3 dimensional of shape** (slices, rows, columns) where the number of rows and columns is 1 and the number of slices matches the intensity array.

metadata: pyfits metadata object, optional The primary FITS metadata describing the data. Although optional, it is strongly recommended that a metadata describing the data be provided. The input subarray mode will be extracted from this metadata. If None, the metadata is ignored and the input subarray mode is assumed to be 'FULL'

intensity_metadata: pyfits metadata object, optional FITS metadata contained in the INTENSITY extension. If None, the metadata is ignored.

wavelength_metadata: pyfits metadata object, optional FITS metadata contained in the WAVELENGTH extension. If None, the metadata is ignored.

fringemap: string, optional The name of a file containing a fringe map to be used to modulate the input illumination. If not specified, no fringe map will be used.

readout_mode: string, optional Readout mode, which can be one of the following common options:

- 'SLOW' - 10 samples per readout and defaults of ngroups=10 and nints=1.
- 'FAST' - 1 sample per readout and defaults of ngroups=1 and nints=10.
- 'FASTINTAVG' - same as FAST but with groups of 4 integrations averaged.
- 'FASTGRPAVG' - same as FAST but with groups of 4 groups averaged.

The following unusual options are also available for testing:

- 'FASTGRPGAP' - a non-MIRI mode similar to FAST mode but with 4 frames per group and a gap of 8 frames between each group.
- 'SLOWINTAVG' - same as SLOW but with groups of 4 integrations averaged and default ngroups=1.
- 'SLOWGRPAVG' - same as SLOW but with groups of 4 groups averaged and default ngroups=4.
- 'SLOWGRPGAP' - a non-MIRI mode similar to SLOW mode but with 4 frames per group and a gap of 8 frames between each group.

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the detector properties default.

subarray: string, optional Detector subarray mode for output. This can be one 'FULL', 'MASK1550', 'MASK1140', 'MASK1065', 'MASKLYOT', 'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64' or 'SLITNESSPRISM', etc. 'FULL' is full-frame and the other modes read out portions of the detector as described in the MIRI Operational Concept Document. The simulator can also accept the other subarray modes defined in detector_properties.py for test purposes. If this parameter is not explicitly given it will be obtained from the FITS header of the input file (unless the input data specifies a non-standard subarray). Failing that, it will be obtained from the detector properties default (FULL).

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the frame time obtained from the detector readout mode and subarray. This can be used, for example, to simulate the timing of full frame exposures with subarray-sized data (to save memory). If None, the frame time obtained from the current readout mode and subarray is used.

inttime: float, optional The integration time in seconds to be simulated. This parameter will be ignored if ngroups is provided as well. *NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.*

ngroups: int, optional The number of readout groups. If None, this is derived from the integration time and readout mode. Otherwise the parameter overrides the integration time and sets the number of groups directly. There must be at least one group.

nints: int, optional The number of integrations per exposure. It must be at least 1. If None, the default set by the readout mode is used. The total exposure time is nints x inttime.

temperature: float, optional, default=detector target temperature The temperature of the detector, which will determine the dark current and readout noise.

cosmic_ray_mode: string, optional The cosmic ray environment mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

If this parameter is not explicitly given it will be obtained from the FITS header of the input file. Failing that, it will be obtained from the cosmic ray properties default.

format: string, optional, default='FITSWriter' The kind of file format to be written.

- 'FITSWriter' - emulate the format written by the FITSWriter during MIRI VM and FM tests and read by the DHAS.
- 'level1' - emulate the level 1 FITS format proposed for the JWST DMS data pipeline..

datashape: string, optional, default='cube' The SCI data format to be written.

- 'hypercube' - write the SCI data to a 4 dimensional FITS image with separate columns x rows x groups x integrations dimensions.
- 'cube' - append the groups and integrations to make a 3 dimensional FITS image with columns x rows x (groups and integrations) dimensions.

clobber: bool, optional, default=False Parameter passed to pyfits.HDUlist.writeto

qe_adjust: boolean, optional, default=True A flag that may be used to switch off quantum efficiency adjustment (for example to observe what effects in a simulation are caused by QE).

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

simulate_ref_pixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of reference pixels in the data.

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_badpixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of bad pixels in the data, even if a bad pixel map containing bad pixels is specified in the detector properties.

simulate_dark_current: boolean, optional, default=True A flag that may be used to switch off the addition of dark current (for example to observe what effects in a simulation are caused by dark current).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the output data is in electrons rather than DNs.*

version: string, optional Software version string.

makeplot: boolean, optional, default=False Plotting flag. Activates plotting of data when True.

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator before generating the test data. If not specified, a value of None will be sent, which randomises the seed.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the simulation parameters are out of range. Also raised if the value of a parameter is invalid or inappropriate.

TypeError Raised if any of the simulation parameters are of the wrong type, size or shape.

KeyError Raised if any of the simulation parameters do not contain a recognised keyword.

IndexError Raised if an attempt is made to access beyond the bounds of the data.

IOError Raised if a file cannot be read, interpreted or written.

ImportError A delayed ImportError is raised if there is an attempt to use an optional library (such as matplotlib) which is not available. The software fails immediately if a compulsory library is not available.

AttributeError Raised if simulation methods are executed in the wrong order, meaning that a necessary attribute is missing. A programming error.

Cosmic ray module (`miri.simulators.scasim.cosmic_ray`)

Description

This module contains all the classes and functions for simulating cosmic ray events. The module contains two classes. The CosmicRayEnvironment class simulates the cosmic ray environment and is programmed with parameters such as the expected cosmic ray flux and energy distribution. The best simulation can be obtained by reading the library of cosmic ray events published by the STScI (M.Robberto). If the cosmic ray library is not available, the CosmicRayEnvironment class can also generate cosmic ray events at random. Each cosmic ray event is described using the CosmicRay class.

Note that the frequency of cosmic ray events is determined both by the properties of the cosmic rays, such as the expected flux, and by the size of the detector pixels and amplifiers, which determines the target area. The thicker pixels of the MIRI detectors produce longer cosmic ray trails.

Objects

class `miri.simulators.scasim.cosmic_ray.`**`CosmicRay`**(*energy*, *target*, *target_coords*, *hit_map*, *nucleon=''*, *verbose=2*)

Class CosmicRay - Describes the properties and the effect of a single cosmic ray event on the MIRI SCA.

Parameters

energy: float The total energy of the cosmic ray event in MeV.

target: string The target hit by the cosmic ray event. Either 'DETECTOR' or 'AMPLIFIER'.

target_coords: int or tuple of 2 ints

- If target is 'DETECTOR', the coordinates of the detector pixel hit by the cosmic ray as (row, column).
- If target is 'AMPLIFIER', the amplifier hit by the cosmic ray.

hit_map: array_like or int

- If target is 'DETECTOR', a map showing the distribution of cosmic ray energy between the central pixel and its neighbours in electrons.
- If target is 'AMPLIFIER', the sum of all the entries in hit_map (which can contain one value) gives the energy absorbed by the amplifier in electrons equivalent.

nucleon: string, optional A description of the nucleon responsible for this cosmic ray.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

`get_electrons`(*scale=1.0*)

Return the total number of electrons released by the cosmic ray scaled by the given factor.

Parameters

scale: float, optional, default=1.0 An optional scale factor to be applied.

Returns

electrons: int The total number of electrons released by the cosmic ray.

`get_energy`()

Return the cosmic ray energy in Mev.

Returns

energy: float The total energy of the cosmic ray event in MeV.

`get_hit_map`(*scale=1.0*)

Returns the hit map scaled by the given factor.

Parameters

scale: float, optional, default=1.0 An optional scale factor to be applied.

Returns

hit_map: array_like

- •If target is 'DETECTOR', a map showing the distribution of cosmic ray energy between the central pixel and its neighbours in electrons.
- •If target is 'AMPLIFIER', the sum of all the entries in hit_map (which can contain one value) givea single value giving the energy absorbed by the amplifier in electrons equivalent.

`get_nucleon`()

Return the nucleon involved.

Returns

nucleon: string A description of the nucleon responsible for this cosmic ray (which can be a null string).

`get_target`()

Return the target hit by the cosmic ray.

Returns

target: string The cosmic ray target ('DETECTOR' or 'AMPLIFIER').

`get_target_coords`()

Return the target coordinates in detector pixels.

Returns

target_coords: tuple of 1 or 2 ints

- •If target is 'DETECTOR', the coordinates of the detector pixel hit by the cosmic ray as (row, column).
- •If target is 'AMPLIFIER', the amplifier hit by the cosmic ray.

`plot`(*plotfig=None*, *plotaxis=None*, *labels=True*, *withbar=False*, *title=None*, *description=''*)

Plot the hit map associated with a cosmic ray event within the given matplotlib axis. This function can can be used to include a plot of this object in any figure. The plotfig method can be used to create a self-contained plot.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot.

labels: bool, optional Set to False to suppress axis labels. The default is True.

withbar: bool, optional Set to True to add a colour bar. The default is False.

title: string, optional Optional title for the plot. The default is a string describing the CosmicRay object. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

description: string, optional Optional description to be added to the title, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Requires

miritools.miriplot matplotlib.pyplot

class `miri.simulators.scasim.cosmic_ray.`**`CosmicRayEnvironment`**(*cr_flux*, *method*, *energies*, *distribution=None*, *images=None*, *nucleons=None*, *verbose=2*)

Class CosmicRayEnvironment - Simulates the behaviour of the cosmic ray radiation in the JWST environment. Galactic cosmic rays and solar particles are both included without being distinguished.

Parameters

cr_flux: float The expected cosmic ray flux incident on the MIRI SCA in number of events per square micron per second.

method: string The method of cosmic ray generation.

- RANDOM - Generate random cosmic rays based on a probability distribution.
- LIBRARY - Read cosmic ray descriptions from a prepared library of events

energies: float array A vector of cosmic ray energies in MeV. This vector can be used in one of two ways, depending on the generation method:

- RANDOM - A list of likely cosmic ray energies
- LIBRARY - A list of cosmic ray energies read from the library.

distribution: float array, optional A vector giving the cumulative probability distribution of the cosmic ray energies. Used by the RANDOM method only. This array must have the same number of elements as the energies array, and it must contain at least one non-zero value.

images: array_like, optional A data cube containing a series of images for each of the cosmic ray events contained in the energies list. Used by the LIBRARY method only. If not None, this array must have a slice for each element of the energies array.

nucleons: int array, optional A list of nucleon IDs for each of the cosmic ray events contained in the energies list. Used by the LIBRARY method only. If not None, this array must be the same number of elements as the energies array.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only

- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

`generate_event` (*target*, *rows*, *columns*, *namps*)

Generate a single cosmic ray event which hits the detector assembly at a random coordinate.

Parameters

target: string The target hit by the cosmic ray - 'DETECTOR' or 'AMPLIFIER'. When target is 'DETECTOR' a random coordinate is chosen within the total rows and columns specified. When target is 'AMPLIFIER' a random amplifier is chosen within the range 1 to namps.

rows: int When target is 'DETECTOR', the number of detector rows.

columns: int When target is 'DETECTOR', the number of detector columns.

namps: int When target is 'AMPLIFIER', the number of detector amplifiers.

Raises

ValueError Raised if any of the parameters are out of range.

Returns

event: CosmicRay A cosmic ray event

`generate_events` (*rows*, *columns*, *time*, *pixsize*, *namps=0*, *amparea=0.0*)

Generate a list of cosmic ray events which happen while a specified detector is integrating for a specified time. Cosmic rays can hit the detector pixels or the detector amplifiers.

Parameters

rows: int The number of detector rows.

columns: int The number of detector columns.

time: float The integration time in seconds.

pixsize: float The detector pixel size in microns.

namps: int, optional, default=0 The number of detector amplifiers. If zero, no amplifier events will be generated.

amparea: float, optional The mean area of the detector surface occupied by each amplifier in square microns.

Returns

event_list: list of CosmicRay objects A list of CosmicRay objects describing the events which take place.

`hist_electrons` (*plotfig*, *plotaxis*, *nbins=256*, *labels=True*, *title=''*)
Add a histogram of the distribution of released detector electrons to a given matplotlib axis. This function can can be used to include a plot of this object in any figure created by the caller.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot.

nbins: int, optional, default=256 The number of histogram bins to be plotted

labels: bool, optional Set to False to suppress axis labels. The default is True.

title: string, optional Optional title to be shown above the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Returns

plotaxis: matplotlib axis object An updated axis containing the plot.

Requires

miritools.miriplot matplotlib.pyplot

`hist_energies` (*plotfig*, *plotaxis*, *nbins=256*, *labels=True*, *title=''*)
Add a histogram of the distribution of cosmic ray energies to a given matplotlib axis. This function can can be used to include a plot of this object in any figure created by the caller.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot.

nbins: int, optional, default=256 The number of histogram bins to be plotted

labels: bool, optional Set to False to suppress axis labels. The default is True.

title: string, optional Optional title to be shown above the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Returns

plotaxis: matplotlib axis object An updated axis containing the plot.

Requires

miritools.miriplot matplotlib.pyplot

`plot` (*nbins=256*, *description=''*)
Plot the distribution of cosmic ray energies in a self-contained matplotlib figure and display it.

Parameters

nbins: int, optional, default=256 The number of histogram bins to be plotted

description: string, optional Additional description to be shown on the plot, if required.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

miritools.miriplot matplotlib.pyplot

`plot_energies`(*plotfig*, *plotaxis*, *labels=True*, *title=''*)

Add a plot of the distribution of cosmic ray energies to a given matplotlib axis. This function can can be used to include a plot of this object in any figure created by the caller.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot.

labels: bool, optional Set to False to suppress axis labels. The default is True.

title: string, optional Optional title to be shown above the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Returns

plotaxis: matplotlib axis object An updated axis containing the plot.

Requires

miritools.miriplot matplotlib.pyplot

`set_metadata`(*metadata*)

Add cosmic ray keywords to the given MIRI metadata object.

Parameters

metadata: dictionary-like object A keyword-addressable metadata object to which cosmic ray environment keywords should be added.

Returns

metadata: dictionary-like object An updated metadata object.

`set_seed`(*seedvalue=None*)

Set the seed for the numpy random number generator. This function can be used while testing to ensure the set of RANDOM cosmic ray events that follows is well defined.

Parameters

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator. If not specified, a value of None will be sent, which randomises the seed.

Functions

`miri.simulators.scasim.cosmic_ray.`**`load_cosmic_ray_random`**(*cosmic_ray_mode='SOLAR_MIN'*, *verbose=2*)

Create a CosmicRayEnvironment object based on the estimated energy distributions contained in cosmic_ray_properties configuration file This will not be as realistic as an environment based on the STScI cosmic ray library.

Parameters

cosmic_ray_mode: string, optional, default='SOLAR_MIN' The cosmic ray mode to be simulated. Available modes are:

- 'NONE' - No cosmic rays.
- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Returns

A new CosmicRayEnvironment object.

`miri.simulators.scasim.cosmic_ray.`**`load_cosmic_ray_library`**(*filename*, *cosmic_ray_mode='SOLAR_MIN'*, *verbose=2*)

Create a CosmicRayEnvironment object from the information contained in a STScI-format cosmic ray library file.

Parameters

filename: string Name of file containing cosmic ray library.

cosmic_ray_mode: string, optional, default='SOLAR_MIN' The cosmic ray mode to be simulated. Available modes are:

- 'SOLAR_MIN' - Solar minimum
- 'SOLAR_MAX' - Solar maximum
- 'SOLAR_FLARE' - Solar flare (worst case scenario)

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Returns

A new CosmicRayEnvironment object.

`miri.simulators.scasim.cosmic_ray.`**`plot_cosmic_ray_events`**(*event_list*, *plotrows*, *plotcols*)

Plot the hit maps from the first few events contain in a cosmic ray event list.

Parameters

event_list: list of CosmicRay objects A list of cosmic ray events. The first (plotrows*plotcols) events will be plotted.

plotrows: int The number of subplot rows to be used.

plotcols: int The number of subplot columns to be used.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

Configuration files

The file cosmic_ray_properties.py contains configuration information.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""

Module cosmic_ray_roperties - Defines the properties of the cosmic
rays expected when the JWST is at L2.

Source: Massimo Robberto, JWST-STScI-001928, SM-12, "A library of
simulated cosmic ray events impacting JWST HgCdTe detectors",
2 December 2009.

:History:

25 Jul 2010: Created
02 Aug 2010: Added cosmic ray mode 'NONE' for no cosmic rays.
30 Aug 2010: Locate the configuration files using an absolute file path
             so that scasim may be run from any directory. Check for
             files contained in the JWST cr_sim_ramp_fit data directory.
27 Sep 2010: Python environment for windows verified.
06 Oct 2010: Changed to new cosmic ray library for SiAs detectors.
24 Oct 2011: Modified to use new "filesearching" module, which should
             make it easier for users to substitute their own configuration
             files (not quite so important for the cosmic ray smiulations).
25 Oct 2011: Corrected bug in energy distribution for NONE mode.
02 Sep 2013: Make more resilient against cr_ramp_fit package not being
             installed.
24 Feb 2014: Use find_cr_library function to find CR library data files
             rather than searching PYTHONPATH (which takes a long time when
            the software is installed using MIRICLE/ureka).

@author: Steven Beard (UKATC)

"""

import os
import warnings

from miri.miritools.filesearching import find_file_in_path

# Get the path to the folder where the cosmic ray library files have been
# installed.
try:
    import cr_sim_ramp_fit
    cr_sim_path = cr_sim_ramp_fit.__path__[0]
    global_cr_data_path = os.path.join(cr_sim_path, 'cr_data_files')
except ImportError:
    warnings.warn("NOTE: cr_sim_ramp_fit package not installed.")
    cr_sim_path = ''
    global_cr_data_path = ''

def find_cr_library( fileprefix, cr_data_path=global_cr_data_path):
    """

    Find the folder containing files with the given named prefix, first
    in a given search path of directories and (failing that) within the
    PYTHONPATH.
    By default, this function will search for a file within the current
    directory the MIRI module  data directories and then try PYTHONPATH.

    This function is used when, rather than there being a single parameter
    file, there are a whole set of parameter files with a common prefix.

    :Parameters:

    fileprefix: str
        A prefix for the files to be located (for example
        ''CR_SiAs_SUNMIN_'', which will cause all files matching
        ''CR_SiAs_SUNMIN_*.*'' to be located).
    cr_data_path: str
        Top level folder where the CR library data files are installed.

    :Returns:

    fileprefixpath: str
        If matching files are found the full path of the folder containing
        the files plus the prefix is returned (e.g. ''/data/CR_SiAs_SUNMIN_'').
        Returns None if no files are found.

    """
    search_path = os.pathsep + './data'
    if cr_data_path:
        search_path += os.pathsep + cr_data_path

    filename = fileprefix + '*.*'
    firsttry = find_file_in_path( filename, search_path=search_path,
                                  pathsep=os.pathsep, walkdir=True,
                                    path_only=True)
    if firsttry:
        return os.path.join(firsttry, fileprefix)
    else:
        return None


DEFAULT_CR_MODE = 'SOLAR_MIN'

# Cosmic ray flux in number/micron^2/sec incident on the MIRI detectors
# at Solar Minimum, Solar Maximum and during a Solar Flare.
# TODO: Would a GROUND mode be useful? If so, what is the flux and spectrum?
CR_FLUX = {}
CR_FLUX['NONE']        = 0.0
CR_FLUX['SOLAR_MIN']   = 4.8983E-8
CR_FLUX['SOLAR_MAX']   = 1.7783E-8
CR_FLUX['SOLAR_FLARE'] = 3.04683E-5
# The following parameter can be used to adjust the cosmic ray flux
# for testing and fine tuning.
CR_FLUX_MULTIPLIER = 1.0

#
# NOTE: There are three sets of library files, suitable for detectors
# with limiting wavelengths of 1.7, 2.5 and 5.5 microns. The 5.5 files
# are chosen as being the closest simulation to the MIRI detectors
# (although these are thicker).
# TODO: If SCASim is to be extended, choose CR library file by detector type.
#
CR_LIBRARY_FILES = {}
CR_LIBRARY_FILES['NONE'] = None
# Library files for the NIRCAM and NIRSPEC detectors (thickness 5.5 microns).
#CR_LIBRARY_FILES['SOLAR_MIN']   = find_cr_library('CRs_MCD5.5_SUNMIN')
#CR_LIBRARY_FILES['SOLAR_MAX']   = find_cr_library('CRs_MCD5.5_SUNMAX')
#CR_LIBRARY_FILES['SOLAR_FLARE'] = find_cr_library('CRs_MCD5.5_FLARES')
# Library files for the MIRI detectors.
if cr_sim_path:
    CR_LIBRARY_FILES['SOLAR_MIN']   = find_cr_library('CRs_SiAs_SUNMIN_')
    CR_LIBRARY_FILES['SOLAR_MAX']   = find_cr_library('CRs_SiAs_SUNMAX_')
    CR_LIBRARY_FILES['SOLAR_FLARE'] = find_cr_library('CRs_SiAs_FLARES_')
else:
    CR_LIBRARY_FILES['SOLAR_MIN']   = None
    CR_LIBRARY_FILES['SOLAR_MAX']   = None
    CR_LIBRARY_FILES['SOLAR_FLARE'] = None
CR_LIBRARY_FILES['MIN'] = 0
CR_LIBRARY_FILES['MAX'] = 9
# The following parameter can be used to adjust the number of electrons
# released by any given cosmic ray for testing and fine tuning.
CR_ELECTRON_MULTIPLIER = 1.0

CR_NUCLEONS = ('H', 'He', 'C', 'N', 'O', 'Fe', '??')

# A map of the leakage of CR flux from a detector pixel into adjacent
# pixels by capacitative coupling.
# TODO: This is a general detector effect, rather than only applying to cosmic ray hits. It also cau
CR_COUPLING = ((0.0,   0.015, 0.0), \
               (0.015, 0.94,  0.015), \
               (0.0,   0.015, 0.0))
```

```
#
# If the library files are not available, this set of electron distributions
# can be used to generate random cosmic ray events independently (although
# not as realistically). The lists contain (energy,probability), where the
# probabilities are incremental and normalised to a peak of 1.0. (They are
# converted to a cumulative distribution internally by the software.)
#
CR_ELECTRONS = {}
CR_ELECTRONS['NONE'] = \
    (
     (0.0,   1.0),
     (1.0e1, 0.0),
     (1.0e2, 0.0),
     (1.0e3, 0.0),
     (1.0e4, 0.0),
     (1.0e5, 0.0),
     (1.0e6, 0.0),
     (1.0e7, 0.0),
     )
CR_ELECTRONS['SOLAR_MIN'] = \
    (
     (1.0e1, 0.0),
     (1.8e1, 0.0),
     (3.2e1, 0.0),
     (5.6e1, 0.02),
     (1.0e2, 0.05),
     (1.8e2, 0.05),
     (3.2e2, 0.05),
     (5.6e2, 0.07),
     (1.0e3, 0.08),
     (1.8e3, 0.10),
     (3.2e3, 0.20),
     (5.6e3, 0.3),
     (1.0e4, 0.5),
     (1.8e4, 1.0),
     (3.2e4, 0.6),
     (5.6e4, 0.15),
     (1.0e5, 0.08),
     (1.8e5, 0.05),
     (3.2e5, 0.03),
     (5.6e5, 0.01),
     (1.0e6, 0.01),
     (1.8e6, 0.003),
     (3.2e6, 0.001),
     (5.6e6, 0.0),
     (1.0e7, 0.0),
     (1.8e7, 0.0),
     (3.2e7, 0.0),
     (5.6e7, 0.0),
     )
CR_ELECTRONS['SOLAR_MAX'] = \
    (
     (1.0e1, 0.0),
     (1.8e1, 0.0),
     (3.2e1, 0.0),
     (5.6e1, 0.01),
     (1.0e2, 0.02),
     (1.8e2, 0.02),
     (3.2e2, 0.02),
     (5.6e2, 0.03),
     (1.0e3, 0.05),
     (1.8e3, 0.07),
     (3.2e3, 0.15),
     (5.6e3, 0.4),
     (1.0e4, 1.0),
     (1.8e4, 0.25),
     (3.2e4, 0.1),
     (5.6e4, 0.05),
     (1.0e5, 0.05),
     (1.8e5, 0.05),
     (3.2e5, 0.05),
     (5.6e5, 0.01),
     (1.0e6, 0.01),
     (1.8e6, 0.005),
     (3.2e6, 0.001),
     (5.6e6, 0.0),
     (1.0e7, 0.0),
     (1.8e7, 0.0),
     (3.2e7, 0.0),
     (5.6e7, 0.0),
     )
CR_ELECTRONS['SOLAR_FLARE'] = \
    (
     (1.0e2, 0.0),
     (1.8e2, 0.001),
     (3.2e2, 0.001),
     (5.6e2, 0.03),
     (1.0e3, 0.05),
     (1.8e3, 0.06),
     (3.2e3, 0.06),
     (5.6e3, 0.07),
     (1.0e4, 0.1),
     (1.8e4, 0.12),
     (3.2e4, 0.25),
     (5.6e4, 0.6),
     (1.0e5, 1.0),
     (1.8e5, 0.6),
     (3.2e5, 0.5),
     (5.6e5, 0.37),
     (1.0e6, 0.18),
     (1.8e6, 0.07),
     (3.2e6, 0.04),
     (5.6e6, 0.03),
     (1.0e7, 0.02),
     (1.8e7, 0.01),
     (3.2e7, 0.0),
     (5.6e7, 0.0),
     )


if __name__ == '__main__':
    print "NOTE: The CosmicRayProperties module is supposed to be " \
        "imported by another module, not run as a main program."
    print "The following cosmic ray properties are defined:"
    print "CR_FLUX\n-------"
    for key in CR_FLUX:
        print "%16s = %s" % (key, CR_FLUX[key])
    print "CR_LIBRARY_FILES\n----------------"
    for key in CR_LIBRARY_FILES:
        print "%16s = %s" % (key, CR_LIBRARY_FILES[key])
    print "CR_NUCLEONS\n--------------"
    print CR_NUCLEONS
    print "CR_COUPLING\n--------------"
    for row in CR_COUPLING:
        print "(%.3f, %.3f, %.3f)" % row
    print "CR_ELECTRONS\n--------------"
    for key in CR_ELECTRONS:
        for row in CR_ELECTRONS[key]:
            str = key
            str += ": Relative probability of %.0f electrons is %f" % row
            print str
    print "Finished."
```

Data maps module (`miri.simulators.scasim.data_maps`)

Description

The data maps module manages various kinds of input data. It contains functions for reading and writing the "SCA format" FITS files and an IlluminationMap class for describing and processing the detector illumination. Central to the IlluminationMap class is a method for combining intensity and wavelength information and returning a single illumination flux. The data maps module also contains classes for managing calibration data.

Objects

class `miri.simulators.scasim.data_maps.`**`DataMap`**(*name*, *unit=''*, *metadata=None*)

Class DataMap - A generic base class describing a map used by the MIRI software (e.g. an illumination map, bad pixel map or calibration data). This class defines only metadata.

Parameters

name: string Name of the data being stored.

unit: string, optional The units of the data being stored. The default is an empty string.

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

`plot`(*clip=1.0*, *description=''*)

Plot one or more images from the data map in a single figure.

Parameters

clip: float, optional, default=1.0 Clip the upper limit of the data plot by this factor. The default of 1.0 means clipping at 100% (i.e. no clipping).

description: string, optional Additional description to be shown on the plot, if required.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

class `miri.simulators.scasim.data_maps.`**`IlluminationMap`**(*intensity*, *wavelength=None*, *metadata=None*, *intensity_metadata=None*, *wavelength_metadata=None*)

Class IlluminationMap - Describes the illumination falling on a detector as a combination of intensity and wavelength.

Parameters

intensity: array_like An array describing the illumination reaching the detector pixels (in photons per second). The array can be 2 dimensional of shape (rows, columns) or 3 dimensional of shape (slices, rows, columns). In both cases the number of rows and columns indicates the detector size in pixels. The third dimension is optional and can be used to divide the illumination into slices at different wavelengths.

wavelength: array_like, optional An array describing the wavelength of the illumination reaching the detector pixels (in microns). If None, this array is ignored; otherwise the wavelength array must match the intensity array in one of 3 ways:

1. The shapes must match exactly; or
2. The wavelength array is 2 dimensional of shape (rows, columns) which exactly match the number of rows and columns in the intensity array; or
3. The wavelength array is 3 dimensional of shape (slices, rows, columns) where the number of rows and columns is 1 and the number of slices matches the intensity array.

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

intensity_metadata: Metadata object, optional Metadata associated with the INTENSITY data. If None, the metadata is ignored.

wavelength_metadata: Metadata object, optional Metadata associated with the WAVELENGTH data. If None, the metadata is ignored.

See Also

load_illumination_map

Raises

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

`get_illumination`(*usefilter=None*)

Get the integrated illumination across all wavelengths. If more than one slice of illumination data are provided they are integrated together, modified by the filter provided. The result is a 2-D illumination array.

Parameters

usefilter: Filter A Filter object describing the transmission or QE as a function of wavelength to be applied to the illumination data. If None, no filter is applied. See the miritools.Filter documentation for a description of the Filter object.

Returns

illumination: array_like A 2-D array containing the integrated illumination.

`get_illumination_enlarged`(*newshape*, *usefilter=None*, *location=(1, 1)*, *leftcrop=0*, *rightcrop=0*)

Get the integrated illumination across all wavelengths, as in method get_illumination, but enlarge the resulting illumination data to the specified shape and size, placing the old data at the given location within the enlarged data and padding the outside areas with zeros.

Parameters

newshape: tuple of 2 ints The new size and shape for the data (rows, columns)

usefilter: Filter A Filter object describing the transmission or QE as a function of wavelength to be applied to the illumination data. If None, no filter is applied. See the miritools.Filter documentation for a description of the Filter object.

location: tuple of 2 ints, optional, default=(1,1) The location at which to place the bottom, left corner of the old data (row, column). Rows and columns start at 1. If not specified, the old data is placed at the bottom left corner.

leftcrop: int, optional, default=0 The number of cropped columns at the left hand edge. The location coordinates have this number of columns subtracted, and the illumination array is cropped if it is placed within this cropped zone.

rightcrop: int, optional, default=0 The number of cropped columns at the right hand edge. The illumination array is cropped if it is placed within this cropped zone.

Returns

illumination_enlarged: array_like A 2-D array containing the integrated illumination enlarged to the new shape.

`get_illumination_shape`()

Get the shape of the integrated illumination map. Note that this is the shape of the integrated illumination map that would be returned by a call to get_illumination. It isn't the shape of the intensity data array.

Returns

shape: tuple The shape of the illumination map (columns, rows)

`get_illumination_withresponse`(*usefilter=None*, *response=None*)

Get the integrated illumination across all wavelengths, as in method get_illumination, but apply a pixel response function while mapping the illumination onto detector pixels.

NOTE: This implementation only works if the illumination data exactly matches the detector response function over the surface of the detector - i.e. if the illuminated portion of the detector is N x M pixels and the response function is R x R, the super-sampled illumination data must be an array of shape (NxR) x (MxR), and this function will return an array of N x M.

Parameters

usefilter: Filter A Filter object describing the transmission or QE as a function of wavelength to be applied to the illumination data. If None, no filter is applied. See the miritools.Filter documentation for a description of the Filter object.

response: array_like, optional A 2-D array containing a pixel response function. If the array has shape R x S it subdivides each pixel into a R x S grid and contains the response of the pixel within each subdivision. If the pixels are square then this response array should also be square (R=S). It is assumed every pixel's response is modified by the same response function across its surface.

Returns

illumination_withresponse: array_like A 2-D array containing the integrated illumination with a pixel response applied, reduced to the same size as the detector.

`plot` (*clip=1.0*, *description=''*)

Plot one or more images of the illumination data in a single figure.

Parameters

clip: float, optional, default=1.0 Clip the upper limit of the intensity plot by this factor. The default of 1.0 means clipping at 100% (i.e. no clipping).

description: string, optional Additional description to be shown on the plot, if required.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`plot_map` (*usefilter=None*, *labels=True*, *withbar=True*, *description=''*)

Plot the illumination map generated from the illumination data when combined with a given filter.

Parameters

usefilter: Filter A filter object describing the transmission or QE as a function of wavelength to be applied to the illumination data. If None, no filter is applied.

labels: bool, optional Set to False to suppress axis labels. The default is True.

withbar: bool, optional Set to False to suppress the colour bar. The default is True.

description: string, optional Additional description to be shown on the plot, if required.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

save (*filename*, *clobber=False*)
Save a detector illumination map to a FITS file.

Note that the arrays appear to be transposed when written to the FITS file, so that an array of shape (slices, rows, columns) is written in the FITS file with NAXIS1=columns, NAXIS2=rows, NAXIS3=slices.

Parameters

filename: string The name of the file to be created.

clobber: bool, optional Parameter passed to pyfits.HDUlist.writeto

Requires

pyfits

class `miri.simulators.scasim.data_maps.`**`DarkMap`**(*dark_map*, *metadata=None*, *dk_metadata=None*)
Class DarkMap - Contains a map describing the variation in dark current over the detector surface, together with a map of hot pixels.

Parameters

dark_map: array_like An array of integer flags describing the health of the detector pixels. The allowed values are defined in the "flagvalues" parameter but the defaults are 0=GOOD, 1=BAD as described below.

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

dk_metadata: Metadata object object, optional The metadata associated with the SCI data. If None, the metadata is ignored.

See Also

load_dark_map

Raises

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

save (*filename*, *clobber=False*)
Save a dark map to a FITS file.

Note that the arrays appear to be transposed when written to the FITS file, so that an array of shape (slices, rows, columns) is written in the FITS file with NAXIS1=columns, NAXIS2=rows, NAXIS3=slices.

Parameters

filename: string The name of the file to be created.

clobber: bool, optional Parameter passed to pyfits.HDUlist.writeto

Requires

pyfits

class `miri.simulators.scasim.data_maps.`**`FringeMap`**(*fringe_map*, *metadata=None*, *fr_metadata=None*)
Class FringeMap - Contains a map describing the fringe pattern over the detector surface.

Parameters

fringe_map: array_like An array of integer flags describing the health of the detector pixels. The allowed values are defined in the "flagvalues" parameter but the defaults are 0=GOOD, 1=BAD as described below.

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

fr_metadata: Metadata object object, optional The metadata associated with the SCI data. If None, the metadata is ignored.

See Also

load_fringe_map

Raises

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

save (*filename*, *clobber=False*)
Save a fringe map to a FITS file.

Note that the arrays appear to be transposed when written to the FITS file, so that an array of shape (slices, rows, columns) is written in the FITS file with NAXIS1=columns, NAXIS2=rows, NAXIS3=slices.

Parameters

filename: string The name of the file to be created.

clobber: bool, optional Parameter passed to pyfits.HDUlist.writeto

Requires

pyfits

class miri.simulators.scasim.data_maps.**BadPixelMap** (*bad_pixels*, *flagvalues=(0, 1)*, *flagnames=('GOOD', 'BAD')*, *metadata=None*, *dq_metadata=None*)
Class BadPixelMap - Contains a map describing the health of the detector pixels.

Parameters

bad_pixels: array_like (converted to uint16) An array of integer flags describing the health of the detector pixels. The allowed values are defined in the "flagvalues" parameter but the defaults are 0=GOOD, 1=BAD as described below.

flagvalues: tuple of ints, optional, default=(0,1) A tuple of allowed flag values.

flagnames: tuple of strings, optional, default=('GOOD','BAD') A tuple of health state names corresponding to each of the flag values. Must be the same length as the flagvalues tuple.

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

dq_header: Metadata objectobject, optional Metadata associated with the DQ extension. If None, the metadata is ignored.

See Also

load_bad_pixel_map

Requires

pyfits

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

`get_counts`(*value*)
Get the number of pixels contained in the data map equal to and not equal to the value provided.

Parameters

data: array-like The data array to be examined

value: int The value to be tested

Returned

counts, tuple of 2 ints The number of pixels equal to and not equal to the given value

`merge_map`(*newmap*, *row=0*, *column=0*)
Merge a new map with the existing bad pixel map. This function enables new bad pixels to be added to an existing bad pixel map.

Parameters

newmap: array_like A bad pixel mask to be merged with the one already contained. The new map will only update places where the existing map identifies healthy pixels.

row: int, optional If newmap is smaller than the bad pixel map, the row at which the new map is to be applied. If not specified, the new map is inserted at the first row.

column: int, optional If newmap is smaller than the bad pixel map, the row at which the new map is to be applied.I f not specified, the new map is inserted at the first column.

`plot`(*description=''*)
Plot a bad pixel map in a single figure.

Parameters

description: string, optional Additional description to be shown on the plot, if required.

Global variables

plt: matplotlib pyplot object The top level pyplot object, imported by this module.

Requires

matplotlib.pyplot

Raises

ImportError Raised if the matplotlib plotting library is not available.

`save` (*filename*, *clobber=False*)
Save a bad pixel map to a FITS file.

Note that the arrays appear to be transposed when written to the FITS file, so that an array of shape (slices, rows, columns) is written in the FITS file with NAXIS1=columns, NAXIS2=rows, NAXIS3=slices.

Parameters

filename: string The name of the file to be created.

clobber: bool, optional Parameter passed to pyfits.HDUlist.writeto

Requires

pyfits

Functions

`miri.simulators.scasim.data_maps.identify_illumination_map` (*filename*)
Identify the sensor chip assembly described by the given illumination map FITS file. The identity is inferred either from the SCA_ID keyword, which contains a numerical code, or failing that from the DETECTOR keyword, which can contain MIRIMAGE, MIRIFULONG or MIRIFUSHORT (Tim Grundy) or IM, LW or SW (Jane Morrison).

Parameters

filename: string or tuple of strings The name(s) of the file(s) to be opened.

Returns

detector: string The detector ID (e.g. 'IM', 'LW' or 'SW'). A null string is returned if the identity cannot be determined.

`miri.simulators.scasim.data_maps.load_illumination_map` (*filename*, *ftype='FITS'*, *metadata=None*, *scale=1.0*, *add_wavelength=True*, *wavmin=1.0*, *wavmax=30.0*)
Create an IlluminationMap object from a file of detector illumination data.

Parameters

filename: string or tuple of strings The name(s) of the file(s) to be opened.

ftype: string, optional, default='FITS' The type of file from which to read the data.

- If 'STSCI', the illumination map is read from a standard MIRI illumination data file, based on the STScI data model.
- If 'FITS', intensity and wavelength data are obtained from a single FITS file containing INTENSITY, WAVELENGTH and DIRECTION extensions (with the latter two extensions being optional).
- If 'ASCII', intensity and wavelength data are read from a list of 1, 2 or 3 ASCII files (with the latter two files being optional). Intensity values are multiplied by the given scale factor. ASCII files describing an illumination map define a value for every pixel. They should contain nrows lines, each of which contains ncolumns of blank-separated numbers; i.e. the default format expected by the numpy.loadtxt() function.

- If 'IMAGE' (or 'JPEG'), intensity data are read from a standard image format file (such as JPEG, GIF or PPM) whose contents are converted into a composite image. A simple wavelength image may be added varying between a specified minimum and maximum wavelength, or wavelength data may be ignored. Intensity values are multiplied by the given scale factor.

metadata: dictionary-like object, optional An object containing metadata to be associated with the data. It could be a plain Python dictionary, a Metadata object or a pyFits Header object, as long as it supports keyword operations. (Only used if the input file is ASCII or IMAGE.)

scale: float, optional, default=1.0 An optional scale factor to apply to the intensity data. This is more useful for ftype='IMAGE' data in cases where the data do not contain a photon flux.

add_wavelength: boolean, optional, default=True Add a test wavelength map to the data. This parameter is valid only for ftype='IMAGE' data.

wavmin: float, optional, default=1.0 Minimum wavelength associated with data in microns. This parameter is valid only for ftype='IMAGE' data.

wavmax: float, optional, default=30.0 Maximum wavelength associated with data in microns. This parameter is valid only for ftype='IMAGE' data.

Raises

ValueError Raised if any of the parameters are out of range.

IOError Raised if there is an error opening, reading or interpreting the data from the input file.

ImportError Raised if the Python Imaging Library (PIL) is not available.

Returns

A new IlluminationMap object.

`miri.simulators.scasim.data_maps.`**`check_wavelength`**(*data*, *shape*, *slices*)

Verifies that a wavelength array is compatible with the shape of the illumination array and number of slices.

Parameters

data: array_like The wavelength data array to be checked.

shape: tuple of 2 ints The illumination data shape.

slices: int The number of slices of illumination data included in the intensity array.

Returns

valid: bool True if the array is compatible and False if not compatible.

`miri.simulators.scasim.data_maps.`**`load_dark_map`**(*filename*, *ftype='FITS'*, *window=None*, *metadata=None*)

Create a DarkMap object from a file.

Parameters

filename: string The name of the file to be opened.

ftype: string, optional, default='FITS' The type of file from which to read the data.

- If 'STSCI', the dark map is read from a standard MIRI image file, based on the STScI data model.
- If 'FITS', the dark map is obtained either from the first extension HDU or from the primary HDU in the given FITS file.

- If 'ASCII', the data are read from an ASCII file in the format recognised by the numpy.loadtxt() function. The file should contain nrows lines, each of which contains ncolumns of blank-separated numbers.
- If 'IMAGE' (or 'JPEG'), the data are read from a standard image format file (such as JPEG, GIF or PPM) whose contents are converted into a composite image.

window: tuple of 4 ints, optional A window to be extracted from the dark map. If not specified, the whole dark map will be used. If specified, the parameter should contain (startrow, startcolumn, rowsize, columnsize). Rows and columns start at 0. The window must be smaller than the dark map defined in the file.

metadata: dictionary-like object, optional An object containing metadata to be associated with the data. It could be a plain Python dictionary, a Metadata object or a pyFits Header object, as long as it supports keyword operations. (Only used if the input file is ASCII or IMAGE.)

Raises

ValueError Raised if any of the parameters are out of range.

IOError Raised if there is an error opening, reading or interpreting the data from the input file.

ImportError Raised if the Python Imaging Library (PIL) is not available.

Returns

A new DarkMap object.

`miri.simulators.scasim.data_maps.`**`load_bad_pixel_map`**(*filename*, *ftype='FITS'*, *flagvalues=None*, *flagnames=None*, *window=None*, *metadata=None*)

Create a BadPixelMap object from a file.

Parameters

filename: string The name of the file to be opened.

ftype: string, optional, default='FITS' The type of file from which to read the data.

- If 'FITS', the bad pixel map is obtained either from the first extension HDU or from the primary HDU in the given FITS file.
- If 'ASCII', the bad pixel map is read from an ASCII file. ASCII files describing a bad pixel map only need to define the non-zero pixels. The ASCII file should contain one row per pixel, and each row contains 3 columns:
  - The row number of the pixel (starting at 0).
  - The column number of the pixel (starting at 0).
  - The value to be stored in the pixel at (row, column).

  Pixels can be defined in any order. The size of the bad pixe map is inferred from the range of values contained in the first two columns. The bad pixel map size can be defined explicitly by setting the first line to "rows-1, ncolumns-1, 0". For example, if a data set is 1024 rows and 1032 columns, the first line would be "1023, 1031, 0".
- If 'IMAGE' (or 'JPEG'), bad pixel data are read from a standard image format file (such as JPEG, GIF or PPM) whose contents are converted into a composite image and then thresholded into discrete levels to make the bad pixel mask. This is useful if the image contains a picture of a bad pixel mask cut from a graphic or created using an image editor.

flagvalues: tuple of ints, optional List of flag values contained in the bad pixel mask. If not given, it will be inferred from the data (except for IMAGE data where the flags will be assumed to be 0 or 1).

flagnames: tuple of strings, optional. List of flag values contained in the bad pixel mask. If not given, it will be inferred from the data (except for IMAGE data where the flags will be assumed to be 'GOOD' or 'BAD').

window: tuple of 4 ints, optional A window to be extracted from the bad pixel mask. If not specified, the whole bad pixel mask will be used. If specified, the parameter should contain (startrow, startcolumn, rowsize, columnsize). Rows and columns start at 0. The window must be smaller than the bad pixel mask defined in the file.

metadata: dictionary-like object, optional An object containing metadata to be associated with the data. It could be a plain Python dictionary, a Metadata object or a pyFits Header object, as long as it supports keyword operations. (Only used if the input file is ASCII or IMAGE.)

Raises

ValueError Raised if any of the parameters are out of range.

IOError Raised if there is an error opening, reading or interpreting the data from the input file.

ImportError Raised if the Python Imaging Library (PIL) is not available.

Returns

A new BadPixelMap object.

`miri.simulators.scasim.data_maps.`**`load_dhas_mask`**(*filename*, *makedark=False*, *hotpixelmult=1000.0*, *window=None*)

Create a BadPixelMap object and (optionally) a DarkMap object from a DHAS bad pixel mask file. The DHAS file contains a map of both dead pixels (which are defined in the SCASim bad pixel mask), hot pixels (which are defined in the SCASim dark map) and noisy pixels (which are not used at the moment).

Parameters

filename: string The name of the DHAS file to be opened.

makedark: boolean, optional True if a DarkMap is to be created from the hot pixels.

hotpixelmult: float, optional. The hot pixel multipler; i.e. the multipler for the dark current of the hot pixels contained in the DarkMap object. The default is 1000.0.

window: tuple of 4 ints, optional A window to be extracted from the bad pixel mask. If not specified, the whole bad pixel mask will be used. If specified, the parameter should contain (startrow, startcolumn, rowsize, columnsize). Rows and columns start at 0. The window must be smaller than the bad pixel mask defined in the file.

Raises

ValueError Raised if any of the parameters are out of range.

IOError Raised if there is an error opening, reading or interpreting the data from the input file.

Returns

A tuple containing (BadPixelMap, DarkMap) objects. When markdark=False, DarkMap is returned None.

`miri.simulators.scasim.data_maps.`**`create_test_data`**(*rows*, *columns*, *pattern*, *values*, *metadata=None*, *cloneimage=None*, *add_wavelength=True*, *wavmin=1.0*, *wavmax=30.0*, *seedvalue=None*)

Create an IlluminationMap or BadPixelMap object containing a test pattern, which can be useful for generating artificial dark and flat-field data, for example.

Parameters

rows: int Number of rows

columns: int Number of columns

pattern: string The type of test pattern. CONSTANT - A constant value. useful for flat-field tests. SLOPE - A flat surface of constant slope. Useful for test data. TESTIMAGE - A regular grid of bright images on a constant background. DARKMAP - A dark current multiplier map containing random hot pixels. BADPIXEL - A bad pixel map containing random dead pixels. Other types TBD.

values: float or tuple of floats Values to be used to create the requested type of data. For CONSTANT:

- values contains the constant value

For SLOPE

- values[0] contains the constant
- values[1] contains the row slope
- values[2] contains the column slope

For TESTIMAGE

- values[0] contains the constant background
- values[1] contains the peak brightness
- values[2] contains the row spacing in pixels
- values[3] contains the column spacing in pixels

For DARKMAP

- values[0] contains the minimum dark multipler (normally <=1.0)
- values[1] contains the maximum dark multipler (normally >=1.0)
- values[2] contains the number of random hot pixels
- values[3] contains the hot pixel multipler (normally >1000)

For BADPIXEL

- values contains the number of random dead pixels

metadata: Metadata object, optional The primary metadata describing the data. If None, the metadata is ignored.

cloneimage: array_like, optional A image to be cloned within the TESTIMAGE, DARKMAP or BADPIXEL patterns. If specified, some of the abnormal pixels will be stamped with this image. If not specified only single pixels are modified.

add_wavelength: boolean, optional, default=True Add a test wavelength map to the data. Not valid for DARKMAP or BADPIXEL.

wavmin: float, optional, default=1.0 Minimum wavelength associated with data in microns.

wavmax: float, optional, default=30.0 Maximum wavelength associated with data in microns.

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator before generating the test data. If not specified, a value of None will be sent, which randomises the seed.

Raises

TypeError Raised if any of the parameters are of the wrong type, size or shape.

Returns

For the CONSTANT and SLOPE patterns, a new IlluminationMap object. For the DARKMAP pattern, a new DarkMap object. For the BADPIXEL pattern, a new BadPixelMap object.

Illumination Map

An illumination map describes the detector illumination.

File Format The illumination map is normally contained in an "SCA format" FITS file (as described in the "MIRI SCA Simulator Software Design" document). For test purposes, it can also be read from an ASCII file or JPEG file.

Files must contain either full-frame (1024x1024) data or subarray data. The SCA simulator will read and process oddly-sized data but can only extract subarrays properly from larger data.

FITS File The primary HDU only contains a header with keywords: This primary header contains information about the MIRI simulation which generated the data. With the exception of the file name keywords FILENAME and ORIGFILE (which are updated), this header is copied unchanged to the output file. The subarray mode can be given in a SUBMODE keyord in the primary HDU header.

The FITS file must contain an extension HDU called 'INTENSITY'. This HDU contains an array describing the intensity of the illumination reaching the detector in photons per second per pixel. This array can be a single 2-D intensity map or can be a 3-D series of slices containing several intensity maps.

The FITS file may optionally contain an extension HDU called 'WAVELENGTH' which contains an array describing the wavelength of the light reaching the detector. If provided, the shape of this array must be compatible with the shape of the INTENSITY map - it must be the same size and shape or be easily broadcast onto the intensity array (see the "MIRI SCA Simulator Design Document" for details).

The FITS file can contain an extension HDU called 'DIRECTION'. This HDU is currently ignored.

JPEG File No header is defined. An intensity map can be provided in a JPEG file (which will be converted into a monochromatic image). The JPEG data can be scaled into units of photons per pixel per second by supplying a scale factor. No wavelength information is extracted from the JPEG file.

ASCII File No header is defined. One, two or three ASCII files may be supplied, containing descriptions of the intensity, wavelength and direction arrays. Each ASCII file contains N columns and M rows, which are read into M x N arrays. Only small 2-D arrays can be defined in this way.

Dark Map

A dark map file describes the variation in dark current over the detector surface in terms of a multiplication factor from its nominal value predicted from the detector temperature. A dark map full of 1.0s would indicate no variation. A dark map can also identify hot pixels with an unusually large dark current.

File Format A dark map is normally contained in a FITS format file. For test purposes, it can also be read from an ASCII file or JPEG file.

Files *must* contain full-frame data including reference pixels (i.e. be 1032 columns x 1024 rows in size). Oddly sized data will be rejected by the software.

FITS File The primary HDU only contains a header with keywords: This primary header contains information about where the dark map came from and which detector it should be used with (as indicated by SCA_ID).

The FITS file must have an extension HDU called 'DARKMAP' containing the dark map.

JPEG File No header is defined. A dark map can be provided in a JPEG file (which will be converted into a monochromatic image). This is only meaningful if the data can be scaled to an average of 1.0.

ASCII File No header is defined. An ASCII file may be supplied containing 1032 columns and 1024 rows of numbers describing the dark map. Probably not useful because such a file would be huge.

Bad Pixel Map

A bad pixel map file identifies bad pixels on the detector surface. Good pixels are identified by a 0 and bad pixels by a 1.

File Format A bad pixel map is normally contained in a FITS format file. For test purposes, it can also be read from an ASCII file or JPEG file.

Files *must* contain full-frame data including reference pixels (i.e. be 1032 columns x 1024 rows in size). Oddly sized data will be rejected by the software.

FITS File The primary HDU only contains a header with keywords: This primary header contains information about where the bad pixel map came from and which detector it should be used with (as indicated by SCA_ID).

The FITS file must have an extension HDU called 'DQ' containing the bad pixel map. (Note: SCASim will also accept bad pixel masks with the data contained in the primary HDU, to allow masks created by DHAS to be used.)

JPEG File No header is defined. A bad pixel map can be provided in a JPEG file (which will be converted into a monochromatic image). The "load_bad_pixel_mask" function which converts the image into a bad pixel mask may be given a threshold so that pixels above the threshold are converted to a 1 and all other pixels are converted to a 2. The ability to read a JPEG image may be useful for extracting a bad pixel mask from a published figure (provided the pixel coordinates in the JPEG can be made to match the detector).

ASCII File No header is defined. An ASCII file may be supplied containing 3 columns of numbers which are "row, column, value". The first line in the file must contain "number of rows-1, number of columns-1, 0". A bad pixel mask will be created of this size initialised full of zeros. Subsequent lines in the ASCII file identify the coordinates of bad pixels. Row and column numbers start at 0.

Exposure data module (`miri.simulators.scasim.exposure_data`)

Description

This module manages the exposure data generated by the SCA simulator and contains functions for writing that exposure data to a FITS file in FITSWriter or DMS 'level 1' format. The ExposureData class manages the data generated by one MIRI exposure. The exposure data contains each of the 2-D images read out from the detector for each readout group and each integration belonging to an exposure.

Objects

class `miri.simulators.scasim.exposure_data.`**`ExposureData`**(*rows*, *columns*, *ngroups*, *nints*, *readpatt*, *grpavg=1*, *intavg=1*, *nframes=1*, *groupgap=0*)

Class ExposureData - Manages the simulated exposure data and the level 1 FITS file output.

Parameters

rows: int The number of rows making up the data.

columns: int The number of columns making up the data.

ngroups: int The number of readout groups making up each integration.

nints: int The number of integrations making up each exposure.

readpatt: str Name of detector readout pattern.

grpavg: int, optional, default=1 The number of groups to be averaged to reduce the file size. *NOTE: An error will be reported if ngroups does not divide exactly by grpavg.*

intavg: int, optional, default=1 The number of integrations to be averaged to reduce the file size. *NOTE: An error will be reported if nints does not divide exactly by intavg.*

nframes: int, optional, default=1 The number of frames per group. Normally 1 for MIRI data.

groupgap: int, optional, default=0 The number of dropped frames in between groups.

Requires

pyfits

Raises

ValueError Raised if any of the initialisation parameters are out of range. Also raised if the combination of initialisation parameters will generate too much data.

`add_fits_comment`(*comment*, *hdu_name='PRIMARY'*)

Add a comment to the comment records contained in the metadata associated with a given FITS HDU.

Parameters

comment: str A string containing a comment.

hdu_name: str, optional, default='PRIMARY' The name of the FITS HDU the keyword is associated with. If None, any HDU is matched, but the keyword must be unique within the data structure.

`add_fits_history`(*history*, *hdu_name='PRIMARY'*)

Add a history record to the metadata associated with a given FITS HDU.

Parameters

history: str A string containing a history record.

hdu_name: str, optional, default='PRIMARY' The name of the FITS HDU the keyword is associated with. If None, any HDU is matched, but the keyword must be unique within the data structure.

`get_fits_comments``()`
Locates all the COMMENT metadata and returns them as a list. Comments found in the PRIMARY metadata are listed first.

Parameters

None

Returns

comments: list of list of strings All the comments found in the metadata. Each item in the top level list corresponds to one HDU and each item in the second level list corresponds to a separate COMMENT entry.

`get_fits_comments_str``()`
Return a structured string containing the comments associated with the data product.

`get_fits_history``()`
Locates all the HISTORY metadata and returns them as a list. History records found in the PRIMARY metadata are listed first.

Parameters

None

Returns

history: list of list of strings All the history found in the metadata. Each item in the top level list corresponds to one HDU and each item in the second level list corresponds to a separate HISTORY entry.

`get_fits_history_str``()`
Return a structured string containing the history records associated with the data product.

`get_fits_keyword`(*keyword*, *hdu_name='PRIMARY'*)
Return the value within the metadata associated with a FITS keyword. Only one value is returned.

Parameters

keyword: str The keyword to be obtained.

hdu_name: str, optional, default='PRIMARY' The name of the FITS HDU the keyword is associated with.

Returns

value: object An object containing the current value associated with the keyword. If multiple occurences are matched, only the first will be returned.

`plot`(*plotstyle='IMAGE'*, *integrations=None*, *groups=None*, *averaged=False*, *recalculate=False*, *clip=(0.0, 1.0)*, *plotsperpage=20*, *description=''*)
Plot the simulated exposure data as a series of frames for each integration and group. A new plot figure is created for each integration. One or more frames can be plotted on each figure.

Parameters

plotstyle: string, optional The plot style, which may be:

- IMAGE - Plot each frame as an image.
- HISTOGRAM - Plot each frame as a histogram.

The default is 'IMAGE'.

integrations: tuple of ints, optional A list of the integrations to be plotted. If not specified, all integrations are plotted.

groups: tuple of ints, optional A list of the groups to be plotted within each integration. If not specified, all groups are plotted.

averaged: boolean, optional, default=False If True, plot what the data would look like after averaging groups and integrations according to grpavg and intavg.

recalculate: boolean, optional, default=False If averaged=True this flag controls whether the averaged data are recalculated if already available.

clip: tuple of 2 floats, optional, default=(0.0, 1.0) The lower and upper clipping limits expressed as a fraction of the data range. The default of (0.0, 1.0) displays the full range of the data.

plotsperpage: int, optional The number of plots to be displayed on each figure. If not specified, the default is the maximum number which can sensibly be fitted. Must be greater than 0.

description: string, optional Additional description to be shown on the plot, if required.

Requires

miritools.miriplot matplotlib.pyplot

Raises

ValueError Raised if any of the parameters are out of range.

IndexError Raised if an attempt is made to plot outside the exposure data area.

plot_ramp (*rows*, *columns*, *stime=1.0*, *tunit=''*, *averaged=False*, *show_ints=False*, *description=''*)

Plot a ramp showing how the signal changes as a function of integration and group at the specified location in the data.

Parameters

rows: int or tuple of ints Either: The row at which the ramp is to be plotted. Or: A list of rows at which ramps are to be plotted.

columns: int or tuple of its. Either: The column at which the ramp is to be plotted. Or: A list of columns at which ramps are to be plotted.

stime: float, optional, default=1.0 The sample time per group in time units.

tunit: string, optional The time unit.

averaged: boolean, optional, default=False If True, plot what the data would look like after averaging groups and integrations according to grpavg and intavg.

description: string, optional Additional description to be shown on the plot, if required.

Requires

miritools.miriplot matplotlib.pyplot

save (*filename*, *fileformat='level1'*, *datashape='cube'*, *clobber=False*, *recalculate=False*)
Creates and writes a level 1 FITS file with the exposure data.

Parameters

filename: string The name of the file to be created.

fileformat: string, optional, default='FITSWriter' The kind of file format to be written.

- 'FITSWriter' - emulate the format written by the FITSWriter during MIRI VM and FM tests and read by the DHAS.
- 'level1' - emulate the level 1 FITS format used by the JWST DMS data pipeline.
- 'level2' - write simple slope data.

datashape: string, optional, default='cube' The SCI data format to be written.

- 'hypercube' - write the SCI data to a 4 dimensional FITS image with separate columns x rows x groups x integrations dimensions.
- 'cube' - append the groups and integrations to make a 3 dimensional FITS image with columns x rows x (groups and integrations) dimensions.

clobber: bool, optional, default=False Parameter passed to pyfits.HDUlist.writeto

recalculate: boolean, optional, default=False If True the averaged data are recalculated even if already available.

Raises

IOError Raised if the output file could not be written.

set_data_header (*header*)
Set the header for the SCI data extensions. Any existing header is deleted.

Parameters

header: pyfits header object The SCI data extension FITS header.

set_exposure (*data*)
Define the entire SCI data array in one go.

Parameters

data: array_like (converted to uint32) An array containing the SCI data. It's size must match the number of rows, columns, groups and integrations defined when the ExposureData object was created.

Raises

TypeError Raised if the data array has the wrong type, size or shape.

set_fits_keyword (*keyword*, *value*, *hdu_name='PRIMARY'*)
Set the metadata associated with a FITS keyword to a given value.

Parameters

keyword: str The keyword to be updated. The keyword must be unique.

value: object The value to set the metadata to.

hdu_name: str, optional, default='PRIMARY' The name of the FITS HDU the keyword is associated with.

`set_group`(*data*, *group*, *intg*)
Add a new readout to the SCI data array.

Parameters

data: array_like (converted to uint32) An array containing the readout data to be added to exposure SCI data. It must have the same shape as one frame of the exposure SCI data (i.e. a 2-D array with the same number of rows and columns specified in the ExposureData constructor).

group: int The group to which the data readout belongs.

intg: int The integration to which the data readout belongs.

Raises

TypeError Raised if the data array has the wrong type, size or shape.

`set_header`(*header*)
Set the primary FITS header. Any existing header is deleted.

Parameters

header: pyfits header object The primary FITS header.

`slope_data`(*grptime=1.0*, *diff_only=False*)
Generate a copy of the SCIdata array where all groups have been combined to make slope data.

NOTE: This function is very inefficient!

Parameters

grptime: float, optional, default=2.785s The time interval between groups (which determines the time axis for the slope calculation).

diff_only: bool, optional, default=False If True, implement a quick and dirty estimate of the slope by subtracting the last frame from the first.

Returned

slope_data: array_like float32 The SCIdata array converted to slopes.

`statistics`(*integration=None*, *group=-1*, *rowstart=None*, *rowstop=None*, *colstart=None*, *colstop=None*, *clip=3.0*, *verbose=False*)
Returns a string describing the mean, standard deviation and other useful statistics for a portion of the exposure data.

Parameters

integration: int, optional, default=all integrationsn The integration to be analysed. If None, all integrations are analysed. -1 means last integration.

group: int, optional, default=last group The group to be analysed. If None, all groups are analyzed. -1 means last group.

rowstart: int, optional, default=all rows The first row to be analysed.

rowstop: int, optional, default=all rows The last row to be analysed.

colstart: int, optional, default=all columns The first column to be analysed.

colstop: int, optional, default=all columns The last column to be analysed.

clip: float, optional, default=3.0 The number of standard deviations outside of which to clip the data to calculate a second pass mean.

verbose: bool Set True for verbose output.

Returned

stats: str String describing the exposure data statistics.

Functions

`miri.simulators.scasim.exposure_data.`**`load_exposure_data`**(*filename*)

Create an ExposureData object from an existing level 1 FITS file. NOTE: Data that has been averaged before being written to a file cannot be unaveraged when read back.

Parameters

filename: string or tuple of strings The name(s) of the file(s) to be opened.

Raises

ValueError Raised if any of the parameters are out of range.

IOError Raised if there is an error opening, reading or interpreting the data from the input file.

Returns A new ExposureData object.

File Format

The exposure data is written either to an STScI data model object (the default), a 'FITSWriter' file, or a 'level 1' FITS file, which are formatted as follows:

The primary HDU contains a FITS header. It contains the header keywords (such as OBSERVER, OBJECT, TELESCOP, INSTRUME, etc...) written by the MIRI simulator which created the detector illumination data plus the following keywords describing the SCA simulation (see the miritools.mirikeyword module for a full list of MIRI keywords):

- ORIGFILE: Name of original detector illumination file
- FILENAME: Name of file created by the SCA simulator
- CRMODE: Cosmic ray mode
- GCR_FLUX: Cosmic ray flux (per square micron per second)
- CRMETHOD: Cosmic ray simulation method.
- FPM_ID: Focal Plane Module ID
- FPM: Focal Plane Module name
- SCA_ID: Sensor Chip Assembly ID
- DETECTOR: ASCII mnemonic for SCA_ID
- CHIP: Type of detector chip

- REFPIXEL: Reference pixels present?
- LREFCOL: Number of left reference columns
- RREFCOL: Number of right reference columns
- BREFROW: Number of bottom reference rows
- TREFROW: Number of top reference rows
- WELL: Detector well depth (electrons)
- DARKCURR: Detector dark current (electrons per second)
- READOUT: Detector readout mode
- NSAMPLE: Total number of samples per readout
- NDISCARD: Number of samples discarded before averaging.
- NFRAME; Number of frames per group (normally 1 for MIRI data)
- GROUPGAP: Number of skipped frames between groups
- NGROUP: Number of groups
- NFRAME: Number of frames per group (1 for MIRI)
- GROUPGAP: Number of frames skipped (0 for MIRI)
- GRPAVG: Number of groups averaged
- NINT: Number of integrations
- INTAVG: Number of integrations averaged
- DETROWS: Number of illuminated rows on the detector
- DETCOLS: Number of illuminated columns on the detector
- SUBMODE: Detector subarray mode
- SUBARRAY: Flag set to T for subarray data.
- FIRSTROW: First subarray row
- FIRSTCOL: First subarray column
- SUBROWS: Number of subarray rows
- SUBCOLS: Number of subarray columns
- TSAMPLE: Detector sample time (seconds)
- TFRAME: Detector frame time (seconds)
- TGROUP: Detector group time (seconds)
- ROWRSETS: Width of reset pulse (clocks)
- FRMRSETS: Number of extra frame resets
- INTTIME: Integration time (seconds)
- EXPTIME: Actual exposure time (seconds)
- REQTIME: Requested exposure time (seconds)
- DETTEMP: Detector temperature (K)
- NOISE<n>: Read noise for amplifier <n> (electrons)

- BIAS<n>: Bias level for amplifier <n> (electrons)
- GAINFN<n>: Simulated amp <n> gain/nonlinearity function
- GCONST<n>: Simulated amp <n> gain constant (e)
- GAIN<n>: Gain for amplifier <n> (DN/electron)
- GP2_<n>: Simulated amp <n> 2nd order gain, if used (DN/e^2)
- GP3_<n>: Simulated amp <n> 3rd order gain, if used (DN/e^3)
- GP4_<n>: Simulated amp <n> 4th order gain, if used (DN/e^4)
- MAXDN<n>: Saturation level for amplifier <n> (DN)

HISTORY and COMMENT records are also written, describing the version number of the software creating the data. The following header keywords are written if they are not already present in the header:

```
ORIGIN  = 'JWST MIRI consortium' / Data origin
TELESCOP= 'JWST    '             / Name of telescope
INSTRUME= 'MIRI    '             / Name of instrument
```

In 'level 1' format, the exposure data is written to an extension HDU named 'SCI'. In 'FITSWriter' format that same data is written to the primary HDU. There are two alternative formats for the data array:

- A 3-D data cube containing a stack of 2-D detector readout images for each integration and group. The default.
- a 4-D hypercube containing a stack of 3-D cubes for each integration; and each cube containing a stack of 2-D images for each group.

Detector module (`miri.simulators.scasim.detector`)

Description

The detector module contains the DetectorArray class, which (together with the amplifier module) simulates the behaviour of a MIRI sensor chip assembly (SCA: 493, 494 or 495) with focal plane module (FPM: 106, 104 or 105).

Note: Other MULTIACCUM detectors (including non-MIRI detectors) can be added by modifying detector_properties.py

Objects

class `miri.simulators.scasim.detector.`**`DetectorArray`**(*detectorid, rows, columns, temperature, left_columns=4, right_columns=4, bottom_rows=0, top_rows=256, well_depth=250000, particle='electron', time_unit='seconds', simulate_poisson_noise=True, simulate_read_noise=True, simulate_badpixels=True, simulate_dark_current=True, simulate_amp_effects=True, makeplot=False, verbose=2*)

Class DetectorArray - Simulates the behaviour of a MIRI detector which consists of an illuminated central zone with extra dark columns at the left and right edges of the detector. Note that the total size of the detector itself in pixels is:

(left_columns + columns + right_columns columns) x (rows)

In addition, this class simulates the rearrangement of the data into the image format expected within a level 1 FITS file, so the class also supports bottom_rows and top_rows additional rows of reference pixel data at the bottom and/or top of the detector. (For MIRI the reference rows are always at the top, but bottom_rows allows some flexibility for reuse with future detectors.)

Parameters

detectorid: string The detector ID, identifying a particular detector. The MIRI instrument has three detectors: 'IM', 'LW' and 'SW'. NOTE: The 'IC' and 'JPL' detectors are not currently recognised.

rows: int The number of illuminated detector rows.

columns: int The number of illuminated detector columns.

temperature: float The detector temperature in K. This is used to look up the dark current and readout noise.

left_columns: int, optional, default=4 The number of extra dark columns at the left edge of the detector.

right_columns: int, optional, default=4 The number of extra dark columns at the right edge of the detector.

bottom_rows: int, optional, default=0 The number of extra reference rows at the bottom edge of the detector.

top_rows: int, optional, default=256 The number of extra reference rows at the top edge of the detector.

well_depth: int, optional, default=250000 The maximum number of particles that can be contained in each detector pixel.

particle: string, optional, default="electron" The name of the particles being counted by the detector pixels.

time_unit: string, optional, default="seconds" The unit of time measurement.

simulate_poisson_noise: boolean, optional, default=True A flag that may be used to switch off Poisson noise (for example to observe what effects in a simulation are caused by Poisson noise).

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_badpixels: boolean, optional, default=True A flag that may be used to switch off the inclusion of bad pixels in the data, even if a bad pixel map containing bad pixels is specified in the detector properties.

simulate_dark_current: boolean, optional, default=True A flag that may be used to switch off the addition of dark current (for example to observe what effects in a simulation are caused by dark current).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the ratio of DNs to electrons is exactly 1.0*

makeplot: boolean, optional, default=False Plotting flag. Activates plotting of data when True.

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised if any of the initialisation parameters are of the wrong type, size or shape.

KeyError Raised if any of the initialisation parameters do not contain a recognised keyword.

`add_bad_pixel_mask` (*filename*, *ftype='STSCI'*)
Add a bad pixel mask associated with the detector.

Parameters

filename: str The name of the file from which to load the bad pixel mask.

ftype: string, optional, default='STSCI' The type of file from which to read the data ('STSCI', 'FITS', 'ASCII' or 'IMAGE').

`add_dark_map` (*filename*, *ftype='STSCI'*)
Add a dark map associated with the detector.

Parameters

filename: str The name of the file from which to load the dark map.

ftype: string, optional, default='STSCI' The type of file from which to read the data ('STSCI', 'FITS', 'ASCII' or 'IMAGE').

`clock_time`()
Return the readout clock time in seconds.

Parameters

None

Returns

clock_time: float The detector clock time in seconds.

`exposure_time` (*nints*, *ngroups*, *nsample*, *subarray=None*, *frame_time=None*, *nframes=1*, *groupgap=0*)
Calculate the exposure time for a given number of integrations and groups, subarray and readout mode (with the ability to override the frame time).

Parameters

nints: int The number of integrations per exposure. Must be at least 1.

ngroups: int The number of groups per integration. Must be at least 1.

nsample: int The number of samples per readout, derived from the readout mode. Usually 1 or 10. Must be at least 1.

subarray: tuple of 4 ints, optional, default is None The subarray mode from which to determine the frame time. If the frame_time parameter is given explicitly (see below), the subarray parameter is ignored. If None a full frame readout is assumed. Otherwise this parameter should be set to subarray parameters (firstrow, firstcol, subrows, subcolumns). *NOTE: Rows and columns are numbered from 1.*

frame_time: float, optional The detector frame time, in seconds. If specified, this parameter overrides the readout mode and subarray and defines the frame time explicitly. It can be used, for example, to calculate the exposure time when simulating full frame exposures over a subarray-sized subset.

nframes: int, optional, default=1 The number of frames per group. Normally 1 for MIRI data.

groupgap: int, optional, default=0 The number of frames dropped between groups. Normally 0 for MIRI data.

Raises

ValueError Raised if any of the parameters are out of range.

Returns

etime: tuple of 2 floats (exp_time, elapsed_time) where exp_time is the exposure time on signal and elapsed_time the total elasped time. The two are the same unless extra resets are used between integrations.

`frame_time`(*nsample*, *subarray=None*)
Calculate the frame time for a given subarray and readout mode.

Parameters

nsample: int The total number of samples per readout, derived from the readout mode (including any discarded samples). Usually 1 or 10. Must be at least 1.

subarray: tuple of 4 ints, optional, default is None If None a full frame readout is assumed. Otherwise this parameter should be set to subarray parameters (firstrow, firstcol, subrows, subcolumns). *NOTE: Rows and columns are numbered from 1 when describing a subarray.*

Raises

ValueError Raised if any of the parameters are out of range.

Returns

ftime: float Frame time in seconds.

`get_counts`()
Return the current electron count for the detector pixels.

`get_subarray_shape`(*subarray=None*)
Get the shape of the data generated by a subarray mode

Parameter

subarray: tuple of 4 ints, optional, default is None If None a full frame readout is assumed. Otherwise this parameter should be set to subarray parameters (firstrow, firstcol, subrows, subcolumns). *NOTE: Rows and columns are numbered from 1 when describing a subarray.*

Returns

subarray_shape: tuple of 2 ints The shape of the subarray data in pixels (rows, columns)

`hit_by_cosmic_rays`(*cosmic_ray_list*, *nframes=1*)
Simulate a series of cosmic ray hits on the detector.

Parameters

cosmic_ray_list: list of CosmicRay objects A list of cosmic ray events hitting the detector.

nframes: int, optional, default=1 The number of frames per group. Normally 1 for MIRI data, but if greater than 1 this parameter will dilute the effect of a cosmic ray hit (since all the frames belonging to a group are averaged.

`integrate` (*photon_flux*, *time*)
Integrate the detector with a certain amount of flux for a certain length of time

Parameters

photon_flux: array_like The photon flux on which to integrate in photons per time unit. This must be the same shape and size as the illuminated portion of the detector.

time: float The integration time in time units.

`plot` (*description=''*)
Plot the photon count and detector readout in a self-contained matplotlib figure.

Parameters

description: string, optional Additional description to be shown on the plot, if required.

Requires

miritools.miriplot matplotlib.pyplot

Side effects

Note that calling plot has the side effect of calling readout(), which can alter simulated effects that change with the number of readouts made.

`plot_readout` (*plotfig=None*, *plotaxis=None*, *labels=True*, *withbar=True*, *title=''*)
Plot the detector readout within the given matplotlib axis. This function can can be used to include a plot of this object in any figure. The plotfig method can be used to create a self-contained plot.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot.

labels: bool, optional Set to False to suppress axis labels. The default is True.

withbar: bool, optional Set to False to suppress the colour bar. The default is True.

title: string, optional Optional title to be shown above the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Requires

miritools.miriplot matplotlib.pyplot

Side effects

Note that calling plot has the side effect of calling readout(), which can alter simulated effects that change with the number of readouts made.

`readout` (*subarray=None*, *nsamples=1*)
Read out the detector (non destructive)

Parameters

subarray: tuple of 4 ints, optional, default is None If None a full frame readout is assumed. Otherwise this parameter should be set to subarray parameters (firstrow, firstcol, subrows, subcolumns). *NOTE: Rows and columns are numbered from 1.*

nsamples: int, optional, default=1 The total number of readout samples. If greater than 1 this reduces the Poisson noise.

Returns

read_data: array_like uint32 The data array read out from the detector.

reset(*nresets=1*, *new_exposure=False*)
Reset the detector

set_metadata(*metadata*, *subarray=None*)
Add detector keywords to the given MIRI metadata.

Parameters

metadata: dictionary-like object A keyword-addressable metadata object to which detector keywords should be added.

subarray: tuple of 4 ints, optional, default is None If None a full frame readout is assumed. Otherwise this parameter should be set to subarray parameters (firstrow, firstcol, subrows, subcolumns). *NOTE: Rows and columns are numbered from 1 when describing a subarray.*

Returns

metadata: dictionary-like object An updated metadata object.

set_readout_mode(*nsample*, *ndiscard=0*, *nframes=1*)
Defines the SCA detector readout mode parameters which determine the read noise.

Parameters

nsample: int The total number of samples per frame (which affects the read noise). Must be at least 1.

ndiscard: int, optional, default=0 The number of samples discarded during a frame (e.g. the first and last samples are usually discarded when nsample > 2).

nframes: int, optional, default=1 The number of frames per group. Normally 1 for MIRI data, but if greater than 1 this parameter further reduces the read noise for each group.

set_seed(*seedvalue=None*)
Set the seed for the numpy random number generator. This function can be used while testing to ensure the set of random choices that follow are well defined.

Parameters

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator. If not specified, a value of None will be sent, which randomises the seed.

Functions

None.

Configuration files

The file detector_properties.py contains configuration information.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""

Module detector_properties - Defines the properties of the MIRI SCA
detector.

NOTE: Other JWST detectors can be simulated as long as their
properties can be described by the parameters contained here.
The main differences are the material, pixel size and detector
thickness.

NOTE: The SCASim simulation is as good as the data contained in
these properties files. Initially, the files contained estimated
properties or properties derived from VM measurements. For the best
simulation, the properties derived from the most recent FM
measurements should be used.

Each set of properties is stored in a Python dictionary and the
properties belonging to a particular focal plane module can be
looked up using a dictionary of dictionaries.

Sources:

(1) JPL D-25632, MIRI Operational Concept Document, 4 March 2010.
(2) JPL D-46944, MIRI Flight Focal Plane Module End Item Data Package
    (FPM EIDP), 10 May 2010.
(3) UA_MIRI_006, MIRI Test Report, Description of Bad Pixel Mask,
    Version 4, 28 August 2011.

:History:

15 Jul 2010: Created
22 Jul 2010: Pixel size and thickness added.
25 Jul 2010: Well depth increased to 100000 (ref 2).
27 Jul 2010: Subarray options added.
01 Aug 2010: Added CHIP_ID and DEFAULTs.
09 Aug 2010: Renamed. QE data file renamed to qe_measurement.txt
16 Aug 2010: Detector properties compiled into a list, since
             MIRI contains three focal plane modules which can behave
             slightly differently. There are now separate dark current
             and QE measurements for each FPM.
24 Aug 2010: QE and dark current measurements now read from FITS files.
             Detector thickness increased to 470 microns.
30 Aug 2010: Locate the configuration files using an absolute file path
             so that scasim may be run from any directory.
03 Sep 2010: Added COSMIC_RAY_LEAKAGE_FRACTION. According to Mike Ressler,
             a small fraction of cosmic ray events can cause charge
             leakage rather than a charge jump.
             Bad pixel maps added.
27 Sep 2010: Python environment for windows verified.
01 Oct 2010: Dropped groups added to readout modes.
12 Oct 2010: Number of frames per group added to readout modes (always 1
             for MIRI but will allow SCAsim to be extended for other JWST
             detectors).
18 Oct 2010: Group gap for GAP modes changed from 4 to 8 to increase the
             exposure time that can be fitted into a reasonable sized
             output file.
11 Nov 2010: DARK_MAP parameters added.
15 Nov 2010: Mistake in CHIP_ID corrected.
22 Nov 2010: Corrections to FPM and SCA IDs in FITS header.
24 Nov 2010: Added test Subarray modes.
15 Dec 2010: SCA_ID values changed from 104,105,106 to 493,494,495
             (as expected by miri_cube)
07 Jan 2011: ID, SCA_ID and DETECTOR values updated to reflect the
             values reported by Tim Grundy (RAL) on 20 Dec 2010.
             Detector properties classified and looked up by SCA_ID
             rather than by FPM_ID.
10 Mar 2011: Subarray modes MASK1065, MASK1550 and SLITLESSPRISM
             corrected to match definitions in OCD Revision C
             (4 March 2010). Subarray modes LRS3, AXIS64, AXIS128 and
             AXIS256 added for test purposes. Subarray modes TEST32
             and TEST64 removed.
25 Mar 2011: PERSISTENCE parameter added, which can be a linear factor
             or a set of polynomial coefficients.
05 Apr 2011: Pixel response function added (not used yet).
21 Sep 2011: The bad pixel masks and dark maps derived from FM tests (3)
             are now used by default.
05 Oct 2011: PERSISTENCE altered for more realistic simulation of
             persistence effects.
             NOTE: Should FRAME_RESETS be increased to simulate the
             post-FM testing adjustment of detector controller parameters?
             NOTE: Which is the correct detector naming convention:
             Tim Grundy's or  Jane Morrison's?
24 Oct 2011: Modified to use new "filesearching" module, which should
             make it easier for users to substitute their own configuration
             USE_FM_MEASUREMENTS flag removed (too complicated).
             files. Pixel response function removed (never used).
13 Jun 2013: Changed detector and subarray names to match new data model.
25 Oct 2013: Make the primary keyword for determining the detector ID
             'DETECTOR' rather than 'SCA_ID'. Detector properties are
             looked up by detector name rather than sca_id.
24 Feb 2014: Use find_simulator_file function to find auxiliary data files
             rather than searching PYTHONPATH (which takes a long time when
             the software is installed using MIRICLE/ureka).
07 May 2014: Added zeropoint drift, nonlinearity and latency factors.
             Added charge trapping and slope drift factors.
08 May 2014: Charge trapping parameter added.
05 Jun 2014: Removed charge trapping parameters and added slow and fast
             latency parameters. Decay parameters now given as a timescale.
19 Jun 2014: Slow and fast zeropoint drift parameters included.

@author: Steven Beard (UKATC)

"""

from miri.simulators.find_simulator_file import find_simulator_file

#
# MIRI contains three focal plane modules, each of which contains a
# detector with a 1024 x 1024 zone illuminated by the instrument. Each
# focal plane module is identified by a unique Sensor Chip Assembly ID
# and a unique Focal Plane Module ID defined as follows. The SCA ID is
# used in MIRI documentation and the FPM ID is used in the FPM EIDP
# (see email from Tim Grundy, RAL):
#    * For the MIRI imager:
#          SCA 493 containing FPM S/N 106.
#    * For the long wavelength arm of the MIRI MRS:
#          SCA 494 containing FPM S/N 104.
#    * For the short wavelength arm of the MIRI MRS:
#          SCA 495 containing FPM S/N 105.
# Each 1024x1024 pixel detector has 4 extra non-illuminated reference
# columns just off the left and right edges of the illuminated zone.
# In addition, there is a separate bank of non-illuminated reference
# pixels ganged together known as reference outputs. These reference
# outputs are not contiguous with the illuminated zone, but the data
# is rearranged in level 1 FITS files so these reference outputs appear
# as extra rows on top of the normal detector image.
#
# Note that DARK_CURRENT_FILE describes how the dark current varies
# with detector temperature. DARK_MAP describes how the dark current
# varies over the detector surface (including hot pixels which have
# excessive dark current).
#
# The find_simulator_file function searches for a named file within a
# search path of simulator data files(starting with the current working
# directory) and returns the absolute path.
#
# TODO: Replace with the measured FM/JPL persistence effects.
# TODO: Are there too many fudge factors describing detector drifts?
_sca493 = {}
_sca493['SCA_ID'] = 493              # Numerical SCA ID
_sca493['FPM_ID'] = "FPMSN106"       # Unique FPM ID
_sca493['NAME'] = "Sensor Chip Assembly 493 with Focal Plane Module 106"
#_sca493['DETECTOR'] = "MIRIMAGE"     # ASCII SCA ID (Tim Grundy)
_sca493['DETECTOR'] = "IM"           # ASCII SCA ID (Jane Morrison)
_sca493['CHIP'] = 'SiAs'             # Type of detector chip
_sca493['COMMENTS'] = "Describes MIRI FPM S/N 106 detector data with ref pixels"
_sca493['ILLUMINATED_ROWS'] = 1024
_sca493['ILLUMINATED_COLUMNS'] = 1024
_sca493['LEFT_COLUMNS'] = 4     # Reference columns on detector
_sca493['RIGHT_COLUMNS'] = 4    # Reference columns on detector
_sca493['BOTTOM_ROWS'] = 0      # There are no extra rows at the bottom
_sca493['TOP_ROWS'] = 256       # Reference rows in level 1 FITS image
_sca493['PIXEL_SIZE'] = 25.0    # Pixel size in microns
_sca493['THICKNESS'] = 470.0    # Detector thickness in microns
_sca493['WELL_DEPTH'] = 250000 # Well depth in electrons
#_sca493['PERSISTENCE'] = 0.0    # Linear persistence factor (0.0 to 1.0)
_sca493['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca493['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay]
_sca493['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay]
_sca493['ZP_SLOW'] = [50000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca493['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca493['LINEARITY'] = [1.0, 0.0]  # Nonlinearity sensitivity coeffs [const,slope]
_sca493['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca493['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca493['CLOCK_PER_REF'] = 3    # Extra settling cycles for reference pixels
_sca493['FRAME_RESETS'] = 0     # Extra resets between integrations
```

```
_sca493['TARGET_TEMPERATURE'] = 6.7   # Target temperature in K
_sca493['DARK_CURRENT_FILE'] = find_simulator_file('dark_current493.fits')
_sca493['QE_FILE'] = find_simulator_file('qe_measurement493.fits')
# Bad pixel mask and dark map derived from FM data:
_sca493['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_IM_BadPixelMask.fits')
_sca493['DARK_MAP'] = find_simulator_file('MIRI_FM_IM_DarkMap.fits')
# Original made up bad pixel mask and dark map:
#_sca493['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixels493.fits')
#_sca493['DARK_MAP'] = find_simulator_file('dark_map493.fits')

_sca494 = {}
_sca494['SCA_ID'] = 494              # Numerical SCA ID
_sca494['FPM_ID'] = "FPMSN104"       # Unique FPM ID
_sca494['NAME'] = "Sensor Chip Assembly 494 with Focal Plane Module 104"
#_sca494['DETECTOR'] = "MIRIFULONG"   # ASCII SCA ID (Tim Grundy)
_sca494['DETECTOR'] = "LW"           # ASCII SCA ID (Jane Morrison)
_sca494['CHIP'] = 'SiAs'             # Type of detector chip
_sca494['COMMENTS'] = "Describes MIRI FPM S/N 104 detector data with ref pixels"
_sca494['ILLUMINATED_ROWS'] = 1024
_sca494['ILLUMINATED_COLUMNS'] = 1024
_sca494['LEFT_COLUMNS'] = 4     # Reference columns on detector
_sca494['RIGHT_COLUMNS'] = 4    # Reference columns on detector
_sca494['BOTTOM_ROWS'] = 0      # There are no extra rows at the bottom
_sca494['TOP_ROWS'] = 256       # Reference rows in level 1 FITS image
_sca494['PIXEL_SIZE'] = 25.0    # Pixel size in microns
_sca494['THICKNESS'] = 470.0    # Detector thickness in microns
_sca494['WELL_DEPTH'] = 250000 # Well depth in electrons
#_sca494['PERSISTENCE'] = 0.0    # Linear persistence factor (0.0 to 1.0)
_sca494['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca494['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay(s)]
_sca494['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay(s)]
_sca494['ZP_SLOW'] = [50000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca494['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca494['LINEARITY'] = [1.0, 0.0]  # Nonlinearity sensitivity coeffs [const,slope]
_sca494['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca494['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca494['CLOCK_PER_REF'] = 3   # Extra settling cycles for reference pixels
_sca494['FRAME_RESETS'] = 0    # Extra resets between integrations
_sca494['TARGET_TEMPERATURE'] = 6.7   # Target temperature in K
_sca494['DARK_CURRENT_FILE'] = find_simulator_file('dark_current494.fits')
_sca494['QE_FILE'] = find_simulator_file('qe_measurement494.fits')
# Bad pixel mask and dark map derived from FM data:
_sca494['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_LW_BadPixelMask.fits')
_sca494['DARK_MAP'] = find_simulator_file('MIRI_FM_LW_DarkMap.fits')
# Original made up bad pixel mask and dark map:
#_sca494['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixels494.fits')
#_sca494['DARK_MAP'] = find_simulator_file('dark_map494.fits')

_sca495 = {}
_sca495['SCA_ID'] = 495              # Numerical SCA ID
_sca495['FPM_ID'] = "FPMSN105"       # Unique FPM ID
_sca495['NAME'] = "Sensor Chip Assembly 495 with Focal Plane Module 105"
#_sca495['DETECTOR'] = "MIRIFUSHORT" # ASCII SCA ID (Tim Grundy)
_sca495['DETECTOR'] = "SW"           # ASCII SCA ID (Jane Morrison)
_sca495['CHIP'] = 'SiAs'             # Type of detector chip
_sca495['COMMENTS'] = "Describes MIRI FPM S/N 105 detector data with ref pixels"
_sca495['ILLUMINATED_ROWS'] = 1024
```

```
_sca495['ILLUMINATED_COLUMNS'] = 1024
_sca495['LEFT_COLUMNS'] = 4     # Reference columns on detector
_sca495['RIGHT_COLUMNS'] = 4    # Reference columns on detector
_sca495['BOTTOM_ROWS'] = 0      # There are no extra rows at the bottom
_sca495['TOP_ROWS'] = 256       # Reference rows in level 1 FITS image
_sca495['PIXEL_SIZE'] = 25.0    # Pixel size in microns
_sca495['THICKNESS'] = 470.0    # Detector thickness in microns
_sca495['WELL_DEPTH'] = 250000 # Well depth in electrons
#_sca495['PERSISTENCE'] = 0.0    # Linear persistence factor (0.0 to 1.0)
_sca495['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca495['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay]
_sca495['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay]
_sca495['ZP_SLOW'] = [50000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca495['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca495['LINEARITY'] = [1.0, 0.0]  # Nonlinearity sensitivity coeffs [const,slope]
_sca495['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca495['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca495['CLOCK_PER_REF'] = 3    # Extra settling cycles for reference pixels
_sca495['FRAME_RESETS'] = 0     # Extra resets between integrations
_sca495['TARGET_TEMPERATURE'] = 6.7    # Target temperature in K
# The pixel response function is the sensitivity variation across the
# surface of each individual pixel.
_sca495['PIXEL_RESPONSE'] = None      # No pixel response function
_sca495['DARK_CURRENT_FILE'] = find_simulator_file('dark_current495.fits')
_sca495['QE_FILE'] = find_simulator_file('qe_measurement495.fits')
# Bad pixel mask and dark map derived from FM data:
_sca495['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_SW_BadPixelMask.fits')
_sca495['DARK_MAP'] = find_simulator_file('MIRI_FM_SW_DarkMap.fits')
# Original made up bad pixel mask and dark map:
#_sca495['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixels495.fits')
#_sca495['DARK_MAP'] = find_simulator_file('dark_map495.fits')

# Other detector descriptions (e.g. for other JWST instruments) can be
# added here.

# The following constants may be imported by SCA simulator software modules.
#
# Dictionary of known focal plane modules. The detector properties for
# each FPM can be obtained by looking up its unique detector name in this
# dictionary and using the result as another dictionary (i.e. the overall
# data structure is a dictionary of dictionaries).
DETECTORS_DICT = {'IM':_sca493,
                  'LW':_sca494,
                  'SW':_sca495}

#
# Available readout modes. The tuple contains default values for:
# nsample  - number of A/D samples making up each readout.
# discard  - number of A/D samples discarded before averaging.
# nframes  - number of frames averaged per group.
# groupgap - number of frames dropped between groups.
# ngroups  - default number of readout groups per integration.
# nints    - default number of integrations per exposure.
# avggrps  - number of groups averaged to reduce data rate.
# avgints  - number of integrations averaged to reduce data rate.
#                                         g
#                               n    d    n    r    n         a    a
#                              s   i   f   o   g       v   v
#                              a   s   r   u   r   n   g   g
#                              m   c   a   p   o   i   g   i
#                              p   a   m   g   u   n   r   n
#                              l   r   e   a   p   t   p   t
#                              e   d   s   p   s   s   s   s
READOUT_MODE = {}
READOUT_MODE['SLOW'] =       (10,  2,  1,  0, 10,  1,  1,  1)
READOUT_MODE['SLOWINTAVG'] = (10,  2,  1,  0,  1,  4,  1,  4)
READOUT_MODE['SLOWGRPAVG'] = (10,  2,  1,  0,  4,  1,  4,  1)
READOUT_MODE['SLOWGRPGAP'] = (10,  2,  4,  8,  4,  1,  1,  1)
READOUT_MODE['FAST'] =       (1,   0,  1,  0,  1, 10,  1,  1)
READOUT_MODE['FASTINTAVG'] = (1,   0,  1,  0,  1,  4,  1,  4)
READOUT_MODE['FASTGRPAVG'] = (1,   0,  1,  0,  4,  1,  4,  1)
READOUT_MODE['FASTGRPGAP'] = (1,   0,  4,  8,  4,  1,  1,  1)
DEFAULT_READOUT_MODE = 'FAST'

#
# Available subarray options. The tuple contains
# (firstrow, firstcol, subrows, subcolumns)
#
SUBARRAY = {}
SUBARRAY['FULL'] = None
SUBARRAY['MASK1065'] =      (   1,     1,  256,  256)
SUBARRAY['MASK1140'] =      ( 229,     1,  256,  256)
SUBARRAY['MASK1550'] =      ( 452,     1,  256,  256)
SUBARRAY['MASKLYOT'] =      ( 705,     1,  320,  320)
SUBARRAY['BRIGHTSKY'] =     (   1,     1,  512,  864)
SUBARRAY['SUB256'] =        (   1,     1,  256,  608)
SUBARRAY['SUB128'] =        ( 897,     1,  128,  132)
SUBARRAY['SUB64'] =         ( 897,     1,   64,   68)
SUBARRAY['SLITLESSPRISM'] = ( 348,     1,  512,   68)
# The following additional subarray options are used
# for testing and simulation only. The detectors are
# never read out using these modes.
# SUBARRAY['LRS1'] =          (   1,     1, 1024,  420)
# SUBARRAY['LRS2'] =          (   1,   292, 1024,  128)
# SUBARRAY['LRS3'] =          (   1,   292,  512,   64)
# SUBARRAY['AXIS256'] =       ( 384,   388,  256,  256)
# SUBARRAY['AXIS128'] =       ( 448,   452,  128,  128)
# SUBARRAY['AXIS64'] =        ( 480,   484,   64,   64)
# SUBARRAY['TEST64'] =        ( 128,   128,   64,   80)
# SUBARRAY['TEST32'] =        (   8,     8,   32,   32)
# SUBARRAY['RHS256'] =        (   1,   776,  256,  256)
STANDARD_SUBARRAYS = ('FULL', 'MASK1065', 'MASK1140', 'MASK1550', 'MASKLYOT',
                      'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64', 'SLITLESSPRISM')
DEFAULT_SUBARRAY = 'FULL'

#
# This fraction of cosmic ray events will cause charge leakage (negative jump)
# rather than a charge increase.
# NOTE: The negative jumps are more likely to be caused by a cosmic raye strike
# on the readout electronics rather than a true charge leakage.
#
COSMIC_RAY_LEAKAGE_FRACTION = 0.01

if __name__ == '__main__':
    print "NOTE: The DetectorProperties module is supposed to be " \
        "imported by another module, not run as a main program."
    print "The following detector properties are defined:"
    for detid in DETECTORS_DICT:
        print "DETECTOR %s\n-----------------" % detid
        detector = DETECTORS_DICT[detid]
        for key in detector:
            print "%24s = %s" % (key, detector[key])
    print "READOUT_MODE\n--------"
    for key in READOUT_MODE:
        print "%24s = %s" % (key, READOUT_MODE[key])
    print "SUBARRAY\n--------"
    for key in SUBARRAY:
        print "%24s = %s" % (key, SUBARRAY[key])
    print "Finished."
```

Amplifier module (`miri.simulators.scasim.amplifier`)

Description

This module contains the Amplifier class, which simulates the behaviour of the readout amplifiers contained in a MIRI focal plane module.

Note: Other amplifiers, including those used by non-MIRI MULTIACCUM detectors can be added by modifying amplifier_properties.py

Objects

class `miri.simulators.scasim.amplifier.`**`Amplifier`**(*number*, *rowmin*, *rowmax*, *rowstep*, *colmin*, *colmax*, *colstep*, *bias*, *gain*, *read_noise*, *maxdn*, *gaintype='POLYNOMIAL'*, *simulate_read_noise=True*, *simulate_amp_effects=True*, *verbose=2*)

Class Amplifier - Simulates the behaviour of a detector readout amplifier. Each amplifier is responsible for a particular portion of the detector surface and has its own bias, gain, read noise and non-linearity properties. All amplifiers are assumed to share some common electronics which contributes an additional bias level which affects all the amplifiers.

Parameters

number: int Unique amplifier number

rowmin: int The minimum row of pixels for which this amplifier is responsible. If zero or None, the amplifier starts with the first row.

rowmax: int The maximum row of pixels for which this amplifier is responsible. If None, the amplifier ends with the last row.

rowstep: int The row increment for this amplifier. 1 or None means every row starting with rowmin up to rowmax. 2 means every 2nd row starting with rowmin up to rowmax. etc... The 3 row parameters form a numpy slice, rowmin:rowmax:rowstep.

colmin: int The minimum column of pixels for which this amplifier is responsible. If zero or None, the amplifier starts with the first column.

colmax: int The maximum column of pixels for which this amplifier is responsible. If None, the amplifier ends with the last column.

colstep: int The column increment for this amplifier. 1 or None means every column starting with colmin up to colmax. 2 means every 2nd column starting with colmin up to colmax. etc... The 3 column parameters form a numpy slice, colmin:colmax:colstep.

bias: int The amplifier bias level in electrons. A constant level added to all readouts.

gain: float, tuple of floats or numpy data cube of floats Coefficients describing the amplifier gain. The meaning varies according to the gain type (see gaintype parameter). For POLYNOMIAL gain type, the gain tuple contains a list of coefficients in reverse order, i.e.

- gain[-1] is a constant in DN.
- gain[-2] is the linear gain in DN/e.
- gain[-3] is the second order coefficient in DN/e^2
- gain[-4] is the third order coefficient in DN/e^3
- etc...

The linear gain is expected to be the dominant term. For LINEAR gain type, the gain parameter is a single float containing just a linear gain in DN/e. For POLYCUBE gain type, the gain parameter is a data cube defining a separate set of polynomial coefficients for each pixel.

read_noise: float The amplifier read noise in electrons.

maxdn: float The maximum output that this amplifier is capable of generating (in DN). If None the output is unlimited.

gaintype: string, optional, default='POLYNOMIAL' The type of gain function. At present only 'LINEAR', 'POLYNOMIAL' and 'POLYCUBE' are supported. NOTE: POLYCUBE has not yet been fully tested.

simulate_read_noise: boolean, optional, default=True A flag that may be used to switch off read noise (for example to observe what effects in a simulation are caused by read noise).

simulate_amp_effects: boolean, optional, default=True A flag that may be used to switch off amplifier effects, i.e. the bias, gain and non-linearity introduced by each amplifier (for example to observe what effects in a simulation are caused by amplifier effects). *Note that when this flag is False the ratio of DNs to electrons is exactly 1.0*

verbose: int, optional, default=2 Verbosity level. Activates print statements when non-zero.

- 0 - no output at all except for error messages
- 1 - warnings and errors only
- 2 - normal output
- 3, 4, 5 - additional commentary
- 6, 7, 8 - extra debugging information

Raises

ValueError Raised if any of the initialisation parameters are out of range.

TypeError Raised of any of the initialisation parameters are of the wrong type.

`amp_effects_off``()`

Disable amplifier bias, gain and non-linearity. They will remain disabled until enabled with amp_effects_on.

amp_effects_on()

Enable amplifier bias, gain and non-linearity. They will remain enabled until disabled with amp_effects_off.

hit_by_cosmic_ray(*energy*)

Simulate a cosmic ray hit on the amplifier.

Parameters

energy: float The energy of the cosmic ray (in terms of the number of electrons it would have released in the detector pixels).

plot(*plotfig=None*, *plotaxis=None*, *labels=True*, *withbar=True*, *description=''*, ***kwargs*)

Plot the amplifier readout count within the given matplotlib axis. This function can can be used to include a plot of this object in any figure. The plotfig method can be used to create a self-contained plot.

Parameters

plotfig: matplotlib figure object Figure on which to add the plot.

plotaxis: matplotlib axis object Axis on which to add the plot. If an axis is provided, the plot will be created within that axis. Providing an axis allows the caller to control exactly where a plot appears within a figure. If an axis is not provided, a new one will be created filling a new figure.

labels: bool, optional Set to False to suppress axis labels. The default is True.

withbar: bool, optional Set to False to suppress the colour bar. The default is True.

description: string, optional Optional description to be added to the title, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Requires

miritools.miriplot matplotlib.pyplot

plotgain(*xmin=0*, *xmax=100000*, *npoints=512*, *title=None*, *description=''*)

Plot amplifier gain as a function of number of electrons to a self-contained matplotlib figure.

Parameters

xmin: int, optional, default=0 The minimum number of electrons to be plotted.

xmax: int, optional, default=100000 The maximum number of electrons to be plotted.

npoints: int, optional, default=512 The number of points to be plotted.

title: string, optional Optional title for the plot. The default is a string describing the Amplifier object. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

description: string, optional Additional description to be shown on the plot, if required. Note: If too much text is written on a plot it may overlap with other labels; especially when applied to subplots.

Requires

miritools.miriplot matplotlib.pyplot

read_noise_off()

Disable Read noise. It will remain disabled until enabled with read_noise_on.

`read_noise_on()`
Enable Read noise. It will remain enabled until disabled with read_noise_off.

`readout`(*detector_data*, *reset_counter=False*, *removeneg=True*)
Apply the bias, gain and readout noise to the portion of the detector data associated with this amplifier. Other areas of the detector data are unaffected.

NOTE: The detector data will not be complete until the readout method from all the associated amplifiers has been applied to it. The readout counter can be used to keep track of how many times readout effects have been applied to different parts of the detector data.

Parameters

detector_data: array_like The detector data on which to apply the readout. *NOTE: The array is converted to floating point. An integer array can wrap around and generate spurious large values if the read noise makes it go negative.*

reset_counter: bool, optional, default=False Force a reset of the counter recording which areas of the detector have had the amplifier properties applied. Otherwise the counter is reset whenever the detector_data array changes shape.

removeneg: bool, optional, default=True If True, remove negative values from the readout and replace them with zero.

Returns

detector_data: array_like Modified detector data. Same shape as the input data.

`reset()`
Reset all cosmic ray effects.

Parameters

None.

`set_metadata`(*metadata*)
Add amplifier keywords to the given MIRI metadata.

Parameters

metadata: dictionary-like object A keyword-addressable metadata object to which amplifier keywords should be added.

Returns

metadata: dictionary-like object An updated metadata object.

`set_readout_mode`(*nsample*)
Defines the SCA detector readout mode.

Parameters

nsample: int The number of (non-discarded) samples included per readout (which affects the read noise). Must be at least 1.

Raises

ValueError Raised if any of the parameters are out of range.

set_seed(*seedvalue=None*)

Set the seed for the numpy random number generator. This function can be used while testing to ensure the randomised amplifier effects that follow are well defined.

Parameters

seedvalue: int, optional, default=None The seed to be sent to the np.random number generator. If not specified, a value of None will be sent, which randomises the seed.

Functions

`miri.simulators.scasim.amplifier.`**set_reference_level**(*newlevel*)

Sets a new electronic reference level level which affects all amplifiers.

Parameters

newlevel: int A new reference level level in electrons.

`miri.simulators.scasim.amplifier.`**random_level_change**(*maxchange*)

Make a random change to the electronic reference level up to the given limit, but don't let the level go below zero. This simulates random drifts that may take place within the electronics and can be used to verify that the pipeline software can remove such drifts by examining the reference pixels.

Parameters

maxchange: int The maximum change to be applied to the reference level in electrons.

`miri.simulators.scasim.amplifier.`**check_readout**()

Check that the readout counters are consistent after reading out all the amplifiers. All the counters should be the same, otherwise some parts of the detector have been missed or other parts have been overlapped and had the readout parameters applied more than once.

Parameters

None.

Returns

ok: bool True if the readout counters are consistent. False if the readout is unfinished or has overlapped.

Configuration files

The file amplifier_properties.py contains configuration information.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""

Module amplifier_properties - Defines the properties of the MIRI SCA
detector readout amplifiers.

The properties are stored in a list of Python dictionaries.

NOTE: The SCASim simulation is as good as the data contained in
these properties files. Initially, the files contained estimated
```

```
properties or properties derived from VM measurements. For the best
simulation, the properties derived from the most recent FM
measurements should be used.

:History:

15 Jul 2010: Created
20 Jul 2010: Bias and reference levels changed to integer.
             Gain polynomial changed to more sensible coefficients.
             Maximum DN value added to simulate amplifier saturation.
27 Jul 2010: Amplifier area added for cosmic ray target calculation.
16 Aug 2010: There can now be several different sets of amplifiers,
             each associated with a different focal plane module.
             The amplifier list associated with each FPM can now be
             looked up in a dictionary.
24 Aug 2010: Read noise measurements now read from FITS files.
30 Aug 2010: Locate the configuration files using an absolute file path
             so that scasim may be run from any directory.
             Normal and reference amplifiers are now counted per
             detector.
02 Sep 2010: Cosmic ray parameters updated.
27 Sep 2010: Python environment for windows verified.
30 Sep 2010: Added GAINTYPE and reduced gain by a factor of 2.7 (following
             comments by Scott Friedman). What are the real gain values?
07 Jan 2011: Amplifier properties classified and looked up by SCA_ID
             rather than by FPM_ID.
04 Feb 2011: Polynomial gain coefficients reversed to make them compatible
             with the numpy.poly1d function.
03 Mar 2011: Linear gain examples added but commented out.
24 Oct 2011: Modified to use new "filesearching" module, which should
             make it easier for users to substitute their own configuration
             files.
25 Oct 2013: Amplifier properties classified and looked up by DETECTOR
             rather than by SCA_ID.
24 Feb 2014: Use find_simulator_file function to find auxiliary data files
             rather than searching PYTHONPATH (which takes a long time when
             the software is installed using MIRICLE/ureka).

@author: Steven Beard (UKATC)

"""

from miri.simulators.find_simulator_file import find_simulator_file

#
# MIRI contains three focal plane modules, each of which contains a
# sensor chip array. Each MIRI SCA uses 5 amplifiers. 4 of the amplifiers
# are assigned to columns on the detector (including the reference
# columns at the left and right edges) while a 5th amplifier is assigned
# to a bank of reference pixels.
#
# The following kinds of gain function (GAINTYPE) are supported. The
# meaning of the GAIN coefficients varies with GAINTYPE:
#
# LINEAR - A single gain coefficient containing the linear gain.
#
# POLYNOMIAL - A set of polynomial coefficients in the order
# accepted by the numpy.poly1d function, e.g. (2nd order, 1st order,
# zeroth order).
#
# POLYCUBE - A set of polynomial coefficients which vary from pixel
# to pixel. NOT FULLY IMPLEMENTED YET.
#
# The read noise measurement for each amplifier is obtained from a file.
# At the moment all the amplifiers belonging to the same FPM share the
# same file and have the same read noise, but it is possible to specify
# a different read noise for every amplifier (if needed).
#
#
# The find_simulator_file function searches for a named file within a
# search path of simulator data files(starting with the current working
# directory) and returns the absolute path.
#
# TODO: Replace with the measured FM/JPL gain and non-linearity settings.
_amp493_1 = {}
_amp493_1['ID'] = 'SCA493_1'
_amp493_1['NAME'] = "Amplifier 1"
_amp493_1['COMMENTS'] = "Reads every 4th detector column starting at column 0"
_amp493_1['TYPE'] = "illuminated"
_amp493_1['REGION'] = "0"    # Starting column
_amp493_1['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp493_1['BIAS'] = 2        # Bias in electrons
_amp493_1['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp493_1['GAIN'] = (-6.6e-8, 0.1683, 0.0)    # Coefficients for gain function
#_amp493_1['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp493_1['GAIN'] = 0.1683                # Coefficients for gain function
_amp493_1['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp493_1['READ_NOISE_FILE'] = find_simulator_file('read_noise493.fits')
_amp493_1['READ_NOISE_COL'] = 1   # Column number where to find read noise

_amp493_2 = {}
_amp493_2['ID'] = 'SCA493_2'
_amp493_2['NAME'] = "Amplifier 2"
_amp493_2['COMMENTS'] = "Reads every 4th detector column starting at column 1"
_amp493_2['TYPE'] = "illuminated"
_amp493_2['REGION'] = "1"    # Starting column
_amp493_2['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp493_2['BIAS'] = 2        # Bias in electrons
_amp493_2['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp493_2['GAIN'] = (-6.6e-8, 0.1720, 0.0)    # Coefficients for gain function
#_amp493_2['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp493_2['GAIN'] = 0.1683                # Coefficients for gain function
_amp493_2['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp493_2['READ_NOISE_FILE'] = find_simulator_file('read_noise493.fits')
_amp493_2['READ_NOISE_COL'] = 2   # Column number where to find read noise

_amp493_3 = {}
_amp493_3['ID'] = 'SCA493_3'
_amp493_3['NAME'] = "Amplifier 3"
_amp493_3['COMMENTS'] = "Reads every 4th detector column starting at column 2"
_amp493_3['TYPE'] = "illuminated"
_amp493_3['REGION'] = "2"    # Starting column
_amp493_3['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp493_3['BIAS'] = 2        # Bias in electrons
_amp493_3['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp493_3['GAIN'] = (-7.0e-8, 0.1646, 0.0)    # Coefficients for gain function
```

```
#_amp493_3['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp493_3['GAIN'] = 0.1683               # Coefficients for gain function
_amp493_3['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp493_3['READ_NOISE_FILE'] = find_simulator_file('read_noise493.fits')
_amp493_3['READ_NOISE_COL'] = 3  # Column number where to find read noise

_amp493_4 = {}
_amp493_4['ID'] = 'SCA493_4'
_amp493_4['NAME'] = "Amplifier 4"
_amp493_4['COMMENTS'] = "Reads every 4th detector column starting at column 3"
_amp493_4['TYPE'] = "illuminated"
_amp493_4['REGION'] = "3"   # Starting column
_amp493_4['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp493_4['BIAS'] = 2       # Bias in electrons
_amp493_4['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp493_4['GAIN'] = (-6.6e-8, 0.1609, 0.0)   # Coefficients for gain function
#_amp493_4['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp493_4['GAIN'] = 0.1683               # Coefficients for gain function
_amp493_4['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp493_4['READ_NOISE_FILE'] = find_simulator_file('read_noise493.fits')
_amp493_4['READ_NOISE_COL'] = 4  # Column number where to find read noise

_amp493_5 = {}
_amp493_5['ID'] = 'SCA493_5'
_amp493_5['NAME'] = "Amplifier 5"
_amp493_5['COMMENTS'] = "Reads from a separate bank of reference pixels"
_amp493_5['TYPE'] = "reference"
_amp493_5['REGION'] = "top" # Arranged at the top of the data (level 1 FITS)
_amp493_5['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp493_5['BIAS'] = 2       # Bias in electrons
_amp493_5['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp493_5['GAIN'] = (-7.0e-8, 0.1683, 0.0)   # Coefficients for gain function
#_amp493_5['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp493_5['GAIN'] = 0.1683               # Coefficients for gain function
_amp493_5['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp493_5['READ_NOISE_FILE'] = find_simulator_file('read_noise493.fits')
_amp493_5['READ_NOISE_COL'] = 5  # Column number where to find read noise

#-----------------------------------------------------------------------------
_amp494_1 = {}
_amp494_1['ID'] = 'SCA494_1'
_amp494_1['NAME'] = "Amplifier 1"
_amp494_1['COMMENTS'] = "Reads every 4th detector column starting at column 0"
_amp494_1['TYPE'] = "illuminated"
_amp494_1['REGION'] = "0"   # Starting column
_amp494_1['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp494_1['BIAS'] = 2       # Bias in electrons
_amp494_1['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp494_1['GAIN'] = (-6.6e-8, 0.1683, 0.0)   # Coefficients for gain function
#_amp494_1['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp494_1['GAIN'] = 0.1683               # Coefficients for gain function
_amp494_1['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp494_1['READ_NOISE_FILE'] = find_simulator_file('read_noise494.fits')
_amp494_1['READ_NOISE_COL'] = 1  # Column number where to find read noise

_amp494_2 = {}
_amp494_2['ID'] = 'SCA494_2'
_amp494_2['NAME'] = "Amplifier 2"
```

```
_amp494_2['COMMENTS'] = "Reads every 4th detector column starting at column 1"
_amp494_2['TYPE'] = "illuminated"
_amp494_2['REGION'] = "1"   # Starting column
_amp494_2['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp494_2['BIAS'] = 2       # Bias in electrons
_amp494_2['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp494_2['GAIN'] = (-6.6e-8, 0.1720, 0.0)   # Coefficients for gain function
#_amp494_2['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp494_2['GAIN'] = 0.1683               # Coefficients for gain function
_amp494_2['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp494_2['READ_NOISE_FILE'] = find_simulator_file('read_noise494.fits')
_amp494_2['READ_NOISE_COL'] = 2  # Column number where to find read noise

_amp494_3 = {}
_amp494_3['ID'] = 'SCA494_3'
_amp494_3['NAME'] = "Amplifier 3"
_amp494_3['COMMENTS'] = "Reads every 4th detector column starting at column 2"
_amp494_3['TYPE'] = "illuminated"
_amp494_3['REGION'] = "2"   # Starting column
_amp494_3['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp494_3['BIAS'] = 2       # Bias in electrons
_amp494_3['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp494_3['GAIN'] = (-7.0e-8, 0.1646, 0.0)   # Coefficients for gain function
#_amp494_3['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp494_3['GAIN'] = 0.1683               # Coefficients for gain function
_amp494_3['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp494_3['READ_NOISE_FILE'] = find_simulator_file('read_noise494.fits')
_amp494_3['READ_NOISE_COL'] = 3  # Column number where to find read noise

_amp494_4 = {}
_amp494_4['ID'] = 'SCA494_4'
_amp494_4['NAME'] = "Amplifier 4"
_amp494_4['COMMENTS'] = "Reads every 4th detector column starting at column 3"
_amp494_4['TYPE'] = "illuminated"
_amp494_4['REGION'] = "3"   # Starting column
_amp494_4['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp494_4['BIAS'] = 2       # Bias in electrons
_amp494_4['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp494_4['GAIN'] = (-6.6e-8, 0.1609, 0.0)   # Coefficients for gain function
#_amp494_4['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp494_4['GAIN'] = 0.1683               # Coefficients for gain function
_amp494_4['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp494_4['READ_NOISE_FILE'] = find_simulator_file('read_noise494.fits')
_amp494_4['READ_NOISE_COL'] = 4  # Column number where to find read noise

_amp494_5 = {}
_amp494_5['ID'] = 'SCA494_5'
_amp494_5['NAME'] = "Amplifier 5"
_amp494_5['COMMENTS'] = "Reads from a separate bank of reference pixels"
_amp494_5['TYPE'] = "reference"
_amp494_5['REGION'] = "top" # Arranged at the top of the data (level 1 FITS)
_amp494_5['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp494_5['BIAS'] = 2       # Bias in electrons
_amp494_5['GAINTYPE'] = 'POLYNOMIAL'         # Type of gain function
_amp494_5['GAIN'] = (-7.0e-8, 0.1683, 0.0)   # Coefficients for gain function
#_amp494_5['GAINTYPE'] = 'LINEAR'         # Type of gain function
#_amp494_5['GAIN'] = 0.1683               # Coefficients for gain function
_amp494_5['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
```

```
_amp494_5['READ_NOISE_FILE'] = find_simulator_file('read_noise494.fits')
_amp494_5['READ_NOISE_COL'] = 5  # Column number where to find read noise

#-----------------------------------------------------------------------------
_amp495_1 = {}
_amp495_1['ID'] = 'SCA495_1'
_amp495_1['NAME'] = "Amplifier 1"
_amp495_1['COMMENTS'] = "Reads every 4th detector column starting at column 0"
_amp495_1['TYPE'] = "illuminated"
_amp495_1['REGION'] = "0"    # Starting column
_amp495_1['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp495_1['BIAS'] = 2        # Bias in electrons
_amp495_1['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp495_1['GAIN'] = (-6.6e-8, 0.1683, 0.0)    # Coefficients for gain function
#_amp495_1['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp495_1['GAIN'] = 0.1683                # Coefficients for gain function
_amp495_1['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp495_1['READ_NOISE_FILE'] = find_simulator_file('read_noise495.fits')
_amp495_1['READ_NOISE_COL'] = 1  # Column number where to find read noise

_amp495_2 = {}
_amp495_2['ID'] = 'SCA495_2'
_amp495_2['NAME'] = "Amplifier 2"
_amp495_2['COMMENTS'] = "Reads every 4th detector column starting at column 1"
_amp495_2['TYPE'] = "illuminated"
_amp495_2['REGION'] = "1"    # Starting column
_amp495_2['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp495_2['BIAS'] = 2        # Bias in electrons
_amp495_2['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp495_2['GAIN'] = (-6.6e-8, 0.1720, 0.0)    # Coefficients for gain function
#_amp495_2['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp495_2['GAIN'] = 0.1683                # Coefficients for gain function
_amp495_2['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp495_2['READ_NOISE_FILE'] = find_simulator_file('read_noise495.fits')
_amp495_2['READ_NOISE_COL'] = 2  # Column number where to find read noise

_amp495_3 = {}
_amp495_3['ID'] = 'SCA495_3'
_amp495_3['NAME'] = "Amplifier 3"
_amp495_3['COMMENTS'] = "Reads every 4th detector column starting at column 2"
_amp495_3['TYPE'] = "illuminated"
_amp495_3['REGION'] = "2"    # Starting column
_amp495_3['AREA'] = 5.0e5    # Area of amplifier surface in square microns
_amp495_3['BIAS'] = 2        # Bias in electrons
_amp495_3['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp495_3['GAIN'] = (-7.0e-8, 0.1646, 0.0)    # Coefficients for gain function
#_amp495_3['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp495_3['GAIN'] = 0.1683                # Coefficients for gain function
_amp495_3['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp495_3['READ_NOISE_FILE'] = find_simulator_file('read_noise495.fits')
_amp495_3['READ_NOISE_COL'] = 3  # Column number where to find read noise

_amp495_4 = {}
_amp495_4['ID'] = 'SCA495_4'
_amp495_4['NAME'] = "Amplifier 4"
_amp495_4['COMMENTS'] = "Reads every 4th detector column starting at column 3"
_amp495_4['TYPE'] = "illuminated"
_amp495_4['REGION'] = "3"    # Starting column
```

```
_amp495_4['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp495_4['BIAS'] = 2       # Bias in electrons
_amp495_4['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp495_4['GAIN'] = (-6.6e-8, 0.1609, 0.0)    # Coefficients for gain function
#_amp495_4['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp495_4['GAIN'] = 0.1683                 # Coefficients for gain function
_amp495_4['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp495_4['READ_NOISE_FILE'] = find_simulator_file('read_noise495.fits')
_amp495_4['READ_NOISE_COL'] = 4  # Column number where to find read noise

_amp495_5 = {}
_amp495_5['ID'] = 'SCA495_5'
_amp495_5['NAME'] = "Amplifier 5"
_amp495_5['COMMENTS'] = "Reads from a separate bank of reference pixels"
_amp495_5['TYPE'] = "reference"
_amp495_5['REGION'] = "top" # Arranged at the top of the data (level 1 FITS)
_amp495_5['AREA'] = 5.0e5   # Area of amplifier surface in square microns
_amp495_5['BIAS'] = 2       # Bias in electrons
_amp495_5['GAINTYPE'] = 'POLYNOMIAL'          # Type of gain function
_amp495_5['GAIN'] = (-7.0e-8, 0.1683, 0.0)    # Coefficients for gain function
#_amp495_5['GAINTYPE'] = 'LINEAR'          # Type of gain function
#_amp495_5['GAIN'] = 0.1683                 # Coefficients for gain function
_amp495_5['MAX_DN'] = 65535.0 # Maximum DN output by this amplifier
_amp495_5['READ_NOISE_FILE'] = find_simulator_file('read_noise495.fits')
_amp495_5['READ_NOISE_COL'] = 5  # Column number where to find read noise


# Lists of defined amplifiers for each FPM.
_amplist494 = (_amp494_1, _amp494_2, _amp494_3, _amp494_4, _amp494_5)
_amplist495 = (_amp495_1, _amp495_2, _amp495_3, _amp495_4, _amp495_5)
_amplist493 = (_amp493_1, _amp493_2, _amp493_3, _amp493_4, _amp493_5)


# The following constants may be imported by SCA simulator software modules.
#
# Dictionary of known focal plane modules. The list of amplifiers for
# each detector can be obtained by looking up its unique detector name
# in this dictionary (i.e. the overall data structure is a dictionary
# of lists of dictionaries).
AMPLIFIERS_DICT = {'LW':_amplist494,
                   'SW':_amplist495,
                   'IM':_amplist493}

# Record the number of amplifiers reading out the illuminated zone
# of the detector and the number of amplifiers reading out the
# reference pixels (as defined by the amplifier type), for each detector.
N_ILLUM_AMPLIFIERS = {}
N_REF_AMPLIFIERS = {}
for detid in AMPLIFIERS_DICT:
    detector = AMPLIFIERS_DICT[detid]
    _icount = 0
    _rcount = 0
    for amp in detector:
        if amp['TYPE'] == 'illuminated':
            _icount += 1
        else:
            _rcount += 1
    N_ILLUM_AMPLIFIERS[detid] = _icount
    N_REF_AMPLIFIERS[detid] = _rcount

# The initial reference level in electrons and the maximum random
# drift in the reference level expected after each readout.
# (These parameters control how short term drifts in the amplifier
# output are simulated.)
INITIAL_REF_LEVEL = 5
MAX_REF_DRIFT = 3

# The following parameters describe how the amplifiers respond when
# hit by a cosmic ray which would have released NE electrons in the
# detector. The following possible effects are considered:
#
# (1) If the cosmic ray strikes an amplifier during an A/D conversion
# it may cause a glitch in the conversion. A high energy cosmic ray
# might affect the next few conversions made by that A/D and generate
# a streak of bad readings. These A/D glitches affect a single reading
# only and the amplifier returns to normal on the next reading.
#
# The COSMIC_RAY_GLITCH_SENSITIVITY parameter defines the sensitivity
# of the A/D converters to a cosmic ray strike. A strike with an energy
# equivalent to NE electrons will cause a streak of glitches
# COSMIC_RAY_GLITCH_SENSITIVITY x NE pixels long. Set this parameter to
# 0.0 to turn off glitches altogether.
COSMIC_RAY_GLITCH_SENSITIVITY = 1.0e-4 # A glitch for every 10000e energy.
#
# The COSMIC_RAY_GLITCH_IPC parameter defines how much cosmic ray energy
# communicated from one A/D conversion to the next. Set this parameter to
# 0.0 to generate single pixel glitches only or to 1.0 to make streaks
# of glitches with no decay in their effect.
COSMIC_RAY_GLITCH_IPC = 0.1 # Each subsequent glitch has 10% of the effect.
#
# (2) A cosmic ray may heat the amplifier slightly or induce unwanted
# currents that cause a temporary increase in read noise. This increased
# noise might linger for the next few readings but will go away the next
# time the detector is reset.
#
# The COSMIC_RAY_NOISE_SENSITIVITY parameter defines how much the read
# noise is affected. A strike with an energy equivalent to NE electrons
# will increase  the read noise by COSMIC_RAY_NOISE_SENSITIVITY x NE
# electrons. Set this parameter to 0.0 to turn off the read noise effect.
COSMIC_RAY_NOISE_SENSITIVITY = 1.0e-5  # 1e of extra noise per 100000e energy.
#
# The COSMIC_RAY_NOISE_DECAY parameter defines by what factor the excess
# read noise decays with each new reading (assuming an exponential decay).
# Set this parameter to 0.0 to make a cosmic ray affect a single reading
# only, or set it to zero to make the excess noise stay the same until
# the next detector reset.
COSMIC_RAY_NOISE_DECAY = 0.5 # Excess noise reduced by x0.5 with every readout.

if __name__ == '__main__':
    print "NOTE: The AmplifierProperties module is supposed to be " \
        "imported by another module, not run as a main program."
    print "Initial reference level = %d and max drift = %d" % \
        (INITIAL_REF_LEVEL, MAX_REF_DRIFT)
    print "The following amplifiers are defined:"
    for detid in AMPLIFIERS_DICT:
        print "          DETECTOR %s\n---------------------------" % detid
        detector = AMPLIFIERS_DICT[detid]
        print "Detector %s has %d normal and %d reference amplifiers." % \
            (detid, N_ILLUM_AMPLIFIERS[detid],
             N_REF_AMPLIFIERS[detid])
        for amp in detector:
            for key in amp:
                print "%24s = %s" % (key, amp[key])
            print "- - - - - - - - - - - - - - - - - - - - -"
    print "Finished."
```

4.1.7 Measurement Data

miri.scasim comes with configuration data describing the cosmic ray, detector and amplifier properties, plus dark current, read noise and quantum efficiency measurements. The measurement data can be provided in ASCII or FITS format files. These files are normally stored in the scasim/data directory, but if SCASim detects files with the same name stored in the current working directory it will use those in preference.. The ASCII files are useful for storing measurement data cut and pasted from elsewhere, whereas the FITS files are more useful because they can store ancilliary information (such as measurement comments) in their header. The 'miritools.make_measurements_fits'and 'make_qe_fits' scripts may be used to convert the ASCII format data into FITS. The make_bad_pixel_mask script can be used to create a tailored bad pixel mask in FITS format. Test calibration files (including bad pixel maps and dark maps) can be made with the make_sca_calibration script. Some examples are given below:

NOTE: SCAsim is now configured by default to use the bad pixel maps and dark current maps derived from FM data by DHAS (e.g. MIRI_FM_BadPixelMask_XXX.fits and MIRI_FM_DarkMap_XXX.fits, where XXX is IM, SW or LW).

./data/amplifiers/read_noiseXXX.txt An ASCII file containing an amplifier read noise measurement for Sensor Chip Assembly XXX (e.g. 493, 494 or 495). The equivalent FITS file has the same name plus a '.fits' extension.

./data/detector/bad_pixelsXXX.fits A FITS file containing a bad pixel mask (for Sensor Chip Assembly XXX).

./data/detector/dark_currentXXX.txt An ASCII file containing a detector dark current measurement (for Sensor Chip Assembly XXX). The equivalent FITS file has the same name plus a '.fits' extension.

./data/detector/dark_mapXXX.fits A FITS file containing a dark current variation map (for Sensor Chip Assembly XXX).

./data/detector/qe_measurementXXX.txt An ASCII file containing a detector quantum efficiency measurement (for Sensor Chip Assembly XXX). The equivalent FITS file has the same name plus a '.fits' extension.

./data/SCATestInput80x64.fits An example SCA-format FITS file containing an 80x64 pixel detector illumination image.

Read Noise Measurement Example (ASCII format)

```
# MIRI Focal Plane Module FPM S/N 106.
#
# Amplifier read noise in electrons as a function of
# amplifier temperature in K.
#
# Source: These values are estimates based on:
#
# (1) "Focal Plane System Performance Status" by Mike Ressler,
#     presented at the MIRI Test Team meeting - 28 June 2010
# NOTE: Readings were taken from the graphs presented. Please replace
# with the actual values measured. Values at high temperatures are
# extrapolated assuming the noise scales with KT.
#
```

```
# Column 1: Temperature (K)
# Column 2: Read noise for channel 1 (electrons)
# Column 3: Read noise for channel 2 (electrons)
# Column 4: Read noise for channel 3 (electrons)
# Column 5: Read noise for channel 4 (electrons)
# Column 6: Read noise for channel 5 (electrons)
 5.00    25.0    25.0    24.0    26.0    25.0
 5.50    25.0    25.0    24.0    26.0    25.0
 5.75    26.0    26.0    25.0    27.0    26.0
 6.00    26.5    26.5    25.5    27.5    26.5
 6.25    28.0    28.0    27.0    29.0    28.0
 6.50    34.5    34.5    33.5    35.5    34.5
 6.75    24.0    24.0    23.0    25.0    24.0
 7.00    24.5    24.5    23.5    25.5    24.5
 7.25    24.0    24.0    23.0    25.0    24.0
 7.50    23.0    23.0    22.0    24.0    23.0
 7.75    23.1    23.1    22.1    24.1    23.1
 8.00    23.2    23.2    22.2    24.2    23.2
 8.25    24.0    24.0    23.0    25.0    24.0
 8.50    25.0    25.0    24.0    26.0    25.0
 8.75    26.5    28.5    30.5    27.5    26.5
 9.00    27.0    29.0    38.0    28.0    27.0
 9.25    28.0    30.0    45.0    29.0    28.0
 9.50    29.0    31.0    50.0    30.0    29.0
10.00    30.5    32.5    55.0    31.5    30.5
10.50    32.0    34.0    57.5    33.0    32.0
11.00    33.5    35.5    60.0    34.5    33.5
11.50    35.0    37.0    62.5    36.0    35.0
12.00    36.5    38.5    65.2    37.5    36.5
13.00    39.5    41.5    67.7    40.5    39.5
14.00    42.5    44.5    69.0    43.5    42.5
15.00    45.5    47.5    70.0    46.5    45.5
```

Dark Current Measurement Example (ASCII format)

```
# MIRI Focal Plane Module FPM S/N 106.
#
# Detector dark current in electrons per second as a function of
# detector temperature in K.
#
# Sources:
#
# (1) JPL D-46944, "MIRI Flight Focal Plane Module End Item Data Package
#     (FPM EIDP)", 10 May 2010.
# (2) "Focal Plane System Performance Status" by Mike Ressler,
#     presented at the MIRI Test Team meeting - 28 June 2010
# NOTE: Readings were taken from the graphs presented. Please replace
# with the actual values measured. Values at high temperatures are
# extrapolated assuming log(dark current) scales linearly with -1/T.
#
# Column 1: Temperature (K)
# Column 2: Dark current (electrons/s)
 5.00    0.08
 5.25    0.07
 5.50    0.055
 5.75    0.07
 6.00    0.08
 6.25    0.09
 6.50    0.1
 6.75    0.12
 7.00    0.46
 7.25    1.60
 7.50    4.94
 7.75   14.4
 8.00   39.0
 8.25  100.0
 8.50  245.0
 8.75  566.0
 9.00  1.25e3
 9.25  2.64e3
 9.50  5.35e3
 9.75  1.05e4
10.00  1.98e4
10.25  3.64e4
10.50  6.49e4
10.75  1.13e5
11.00  1.91e5
11.25  3.12e5
11.50  5.10e5
11.75  8.08e5
12.00  1.26e6
13.00  6.196e6
14.00  2.433e7
15.00  7.960e7
```

Quantum Efficiency Measurement Example (ASCII format)

```
# MIRI Sensor Chip Assembly 494 + Focal Plane Module FPM S/N 104.
#
# Detector quantum efficiency (as modified by the detector AR coating)
# as a function of wavelength in microns.
#
# Sources:
#
# (1) JPL D-46944, "MIRI Flight Focal Plane Module End Item Data Package
#     (FPM EIDP)", 10 May 2010.
# (2) "Focal Plane System Performance Status" by Mike Ressler,
#     presented at the MIRI Test Team meeting - 28 June 2010
# (3) Data file for MIRI MRS created by Kevin Lindsay and Chris Hanley
#     (STScI) for ETC prototype, Sept 2010.
#
# Column 1: Wavelength (microns)
# Column 2: Quantum efficiency (0-1)
 0.00        0.001
 2.00        0.306
 2.50        0.363
 3.00        0.286
 3.50        0.392
 4.00        0.322
 4.50        0.333
 5.00        0.405
 5.50        0.496
 6.00        0.542
 6.50        0.536
 7.00        0.480
 7.50        0.454
 8.00        0.476
 8.50        0.487
 9.00        0.497
 9.50        0.496
10.00        0.528
10.50        0.539
11.00        0.545
11.50        0.555
12.00        0.574
12.50        0.600
13.00        0.604
13.50        0.597
14.00        0.650
14.40        0.725
14.50        0.725
15.00        0.719
15.50        0.705
16.00        0.672
16.50        0.691
17.00        0.722
17.50        0.688
18.00        0.643
18.50        0.637
19.00        0.605
19.50        0.602
20.00        0.564
20.50        0.576
21.00        0.581
21.50        0.603
22.00        0.590
22.50        0.583
23.00        0.560
23.50        0.527
24.00        0.514
24.50        0.473
25.00        0.422
25.50        0.373
26.00        0.325
26.50        0.268
27.00        0.189
27.50        0.137
28.00        0.084
28.50        0.060
29.00        0.043
29.50        0.028
30.00        0.018
30.50        0.001
31.00        0.001
```

Bad Pixel Specification Example (ASCII format)

```
#
# An example ASCII file defining a bad pixel mask.
# The mask will be 32 x 32 pixels in size and
# contain bad pixels looking like the word "bad".
# Rows and columns are numbered from 0 to 31.
#
31 31  0
 2  2  1
 2  3  1
 2  4  1
 3  2  1
 3  4  1
 3  5  1
 4  2  1
 4  5  1
 5  2  1
 5  3  1
 5  4  1
 5  5  1
 6  2  1
 6  4  1
 6  5  1
 7  2  1
 7  4  1
 8  2  1
 8  3  1
 8  4  1

 2  9  1
 2 15  1
 3  9  1
 3 10  1
 3 14  1
 3 15  1
 4 10  1
 4 14  1
 5 10  1
 5 11  1
 5 12  1
 5 13  1
 5 14  1
 6 11  1
 6 13  1
 7 11  1
 7 13  1
 8 12  1

 2 19  1
 2 20  1
 2 21  1
 3 19  1
 3 22  1
 4 19  1
 4 23  1
 5 19  1
 5 23  1
 6 19  1
 6 23  1
 7 19  1
 7 22  1
 8 19  1
 8 20  1
```

```
8 21  1
```