Science & Technology Facilities Council
UK Astronomy Technology Centre

MIRI Sensor Chip Assembly (SCA) Simulator User Manual

(V2.0)

Steven Beard

UK Astronomy Technology Centre
Royal Observatory
Blackford Hill
Edinburgh
EH9 3HJ

1 INTRODUCTION

MIRI is a mid infrared instrument for the James Web Space Telescope (JWST), [5]. The instrument contains an imager, which can also be used as a coronagraph or low resolution spectrograph (LRS), and a medium resolution spectrograph (MRS). MIRI contains three 1024x1024 pixel detector chips: one installed in the imager and two installed in the MRS.

The data generated by the MIRI instrument can be simulated using a series of instrument simulator applications: for example for example the imager simulator (ImSim) simulates MIRI imager data and the MRS simulator (formerly known as Specsim), [7] and [9], simulates MIRI MRS data. Both parts of the MIRI instrument use the same design of Sensor Chip Assembly (SCA) and the same kind of detector chip. The MIRI Sensor Chip Assembly Simulator (SCASim) provides a common utility which can be used by all MIRI simulators. The role of the SCA simulator is illustrated in Figure 1 below. The MIRI instrument simulator starts with an astronomical target and simulates the telescope and instrument effects. The end result is a description of how the instrument is illuminating each instrument detector, which is saved to a FITS file in a standard format common to all simulators (see the software design document, [1]). The SCA simulator reads that FITS file and finishes the simulation by adding the detector effects.

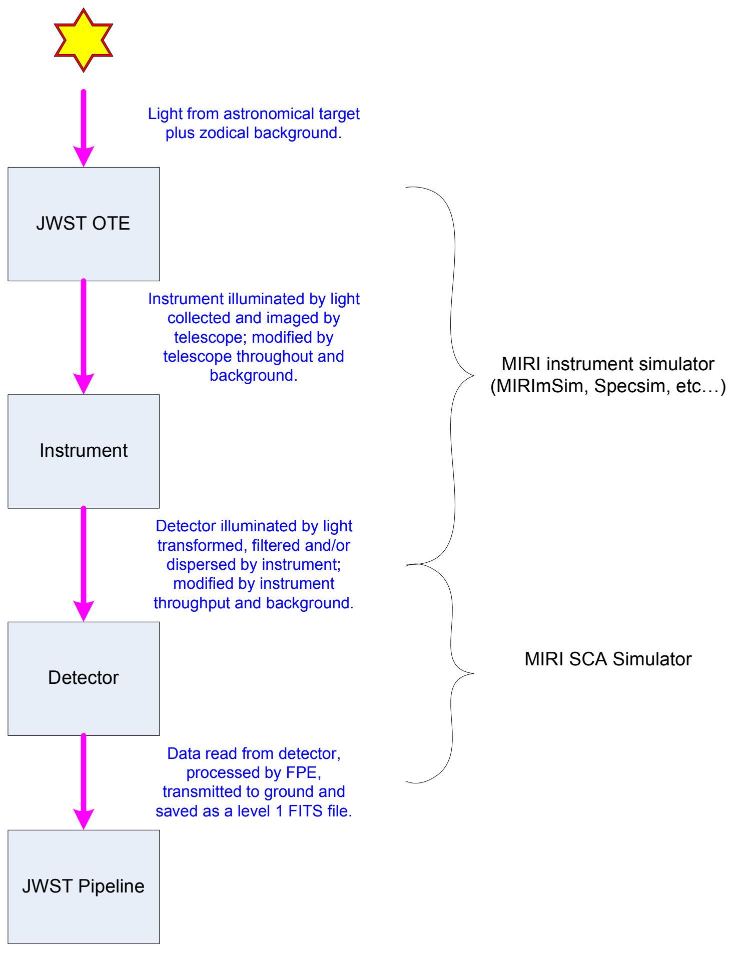
Light from astronomical target plus zodical background.
JWST OTE
Instrument illuminated by light collected and imaged by telescope; modified by telescope throughout and background.
Instrument
Detector illuminated by light transformed, filtered and/or dispersed by instrument; modified by instrument throughput and background.
Detector
Data read from detector, processed by FPE, transmitted to ground and saved as a level 1 FITS file.
JWST Pipeline
MIRI instrument simulator (MIRImSim, Specsim, etc…)
MIRI SCA Simulator

Figure 1: Overview of the Role of the MIRI SCA Simulator

The SCA simulator simulates the following detector effects (described in more detail in section 7 on page 16):

- Simulation of Poisson (shot) noise.
- Simulation of dead pixels and hot pixels.
- Simulation of detector persistence and latency.
- Simulation of zero-point drift.
- Simulation of detector non-linearity.
- Simulation of pixel flat-field.

- Simulation of dark current.
- Simulation of the effect of quantum efficiency as a function of wavelength
- Simulation of read noise as a function of temperature.
- Simulation of reference pixels.
- Simulation of amplifier gain and non-linearity.
- Simulation of detector latency.
- Simulation of MULTIACCUM detector readout modes.
- Simulation of MIRI subarray modes.
- Simulation of cosmic ray hits on the detector.

2 ACRONYMS AND ABBREVIATIONS

ADC	-	Analogue to Digital Converter
GUI	-	Graphical User Interface
FPA	-	Focal Plane Array
FPAP	-	Focal Plane Array Processor
FPE	-	Focal Plane Electronics
FPM	-	Focal Plane Module
FPS	-	Focal Plane System
GUI	-	Graphical User Interface
ICE	-	Instrument Control Electronics
ICDH	-	Integrated Command and Data Handling
IDL	-	Interactive Data language
IFU	-	Integrated Field Unit
LRS	-	Low Resolution Spectrograph
FITS	-	Flexible Image Transport System
JPL	-	Jet Propulsion Laboratory
JWST	-	James Webb Space Telescope
MIRI	-	Mid Infra-Red Instrument
MRS	-	Medium Resolution Spectrograph
MTS	-	MIRI Telescope Simulator
OTE	-	Optical Telescope Element
PSF	-	Point Spread Function
ROI	-	Region Of Interest
SCA	-	Sensor Chip Assembly
SRON	-	Netherlands Institute for Space Research
SSR	-	Solid State Recorder
STScI	-	Space Telescope Science Institute
UKATC	-	United Kingdom Astronomy Technology Centre

For other acronyms see http://www.roe.ac.uk/ukatc/consortium/miri/acronyms.html .

3 REFERENCES

[1] *MIRI Sensor Chip Assembly (SCA) Simulator Software Design Document*, Steven Beard, UKATC, 8 June 2016.

[2] *MIRI SCASim Software Reference Manual*, Steven Beard, UKATC, 7 June 2016.

[3] *MIRI miritools Software Reference Manual*, Steven Beard, 7 June 2016.

[4] MIRICLE - Miri Analysis System in Python, Wim der Meester, 7 June 2016

http://miri.ster.kuleuven.be/bin/view/Internal/MiricleInstallation

[5] JPL D-25632, *MIRI Operations Concept Document (Revision C)*, Christine Chen, Karl Gordon, Scott Friedman and Margaret Meixner, STScI, 4 March 2010

[6] *JWST Calibration Software Online Documentation*, http://stsdas.stsci.edu/jwst/docs/sphinx/

[7] *DHAS miri_sloper Online Documentation*, http://tiamat.as.arizona.edu/dhas/

[8] *JWST MIRI Specsim (V1) User Guide*, Nuria Lorente, UKATC, 30 March 2006.

[9] *Specsim V2 User Guide*, Dennis Kelly and Nuria Lorente, UKATC, 18 June 2009.

[10] *Specsim Programming Guide (V0.2)*, Steven Beard, UKATC, 31 March 2010.

[11] 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

[12] 6. MIRI DFM 308-04.02, *Analysis of the Proton Flux on the MIRI Detector Arrays*, M. Ressler, 3 August 2009.

[13] JWST-STScI-00198, SM-12, *A Library of Simulated Cosmic Ray Events Impacting JWST HgCdTe Detectors*, M. Robberto, 9 March 2010.

[14] MIRI FM Raw Science Data Definition (Issue 1, Draft B), Tim Grundy, 12 January 2011.

[15] Simulation of MIRI Detector Latent Effects with SCASIm, Steven Beard, 25 June 2014.

4 INSTALLATION

NOTE: The easiest way to install the SCA simulator is to install the MIRI software using the MIRICLE installation script, [4]. If you have installed the simulator using MIRICLE, skip to section 4.5 on page 7.

For more information about installing Python software, see the Python installation page on the wiki at http://miri.ster.kuleuven.be/bin/view/Internal/Software/PythonInstall . More general information on Python programming is available at at http://miri.ster.kuleuven.be/bin/view/Internal/Software/PythonProgramming .

4.1 Prerequisites

4.1.1 Python

The SCA simulator is written in Python and is compatible with Python 2.5, 2.6 or 2.7. It requires a computer on which the following additional Python components have been successfully installed:

- numpy
- scipy
- matplotlib (optional)

The easiest way to install both Python and the necessary extras is to install the MIRI software using the MIRICLE script described at http://miri.ster.kuleuven.be/bin/view/Internal/MiricleSoftwareSystem .

4.1.2 STScI Python

The SCA simulator also requires STScI Python utilities to be installed:

- jwst_lib
- astropy.io.fits

See the STScI Python home page at http://www.stsci.edu/resources/software_hardware/pyraf/stsci_python for instructions on how to obtain and install STScI Python. There is also information on pyfits at http://www.stsci.edu/resources/software_hardware/pyfits .

4.1.3 Simple Check

To check whether Python and STScI Python has been installed on your system, try the following Python commands:

```
python
>>> import numpy
>>> print numpy.__version__
>>> import scipy
>>> print scipy.__version__
>>> import astropy.io.fits
>>> print astropy.io.fits.__version__
```

All of these commands must succeed before SCASim can be installed. In particular, if the “setup” commands described in section 4.4 will fail with an error message complaining that module “distutils_hack” is not available this is a symptom that stsci_python is not properly installed. Import errors can be caused either by the software not being installed on your system or by the PYTHONPATH environment variable not pointing to the correct directories (see section 4.4.1 for an example).

The print statements are optional, but they can be used to compare the actual version numbers of the modules installed with the recommended versions. Minor differences are usually ok as long as the version you have installed is more recent but if you run into unexplainable problems try matching the tested version numbers exactly. If there are any compatibility issues with more recent releases of STScI Python please let me know.

4.2 Obtaining the SCA simulator

The SCA simulator (SCASim) can be downloaded from the JWST repository hosted at the STScI using the subversion client (svn). SCASim is part of the MIRI software suite, which can be installed with the following command (executed from the directory where you would like the software to be installed):

```
svn checkout https://aeon.stsci.edu/ssb/svn/jwst/trunk/teams/miri/ ./miri
```

This command will install the software into a new directory named “miri”. If an accurate cosmic ray simulation is important then the JWST prototypes package should also be installed (since it contains the cr_sim_ramp_fit module):

```
svn checkout https://aeon.stsci.edu/ssb/svn/jwst/trunk/prototypes/ ./prototypes
```

Alternatively, the simulator can be installed alongside the entire JWST software suite with the command:

```
svn checkout https://aeon.stsci.edu/ssb/svn/jwst/trunk ./jwst
```

in which case the MIRI software will be installed into the “jwst/teams/miri” directory.

4.3 Folder and file layout

After the MIRI software has been installed, the overall layout of the folders and files will look like Figure 2. The top level “miri” folder contains a folder for each software package. The folder “miritools” contains a package of general purpose tools (documented separately) and the SCA simulator may be found within the folder “scasim”, which contains the following subfolders:

- **data** – contains data files and configuration data
- **doc** – contains documentation
- **lib** – contains Python source code modules
- **scripts** – contains Python main programs and scripts
- **tests** – contains unit tests

The layout should be familiar to anyone who has used STScI Python utilities.

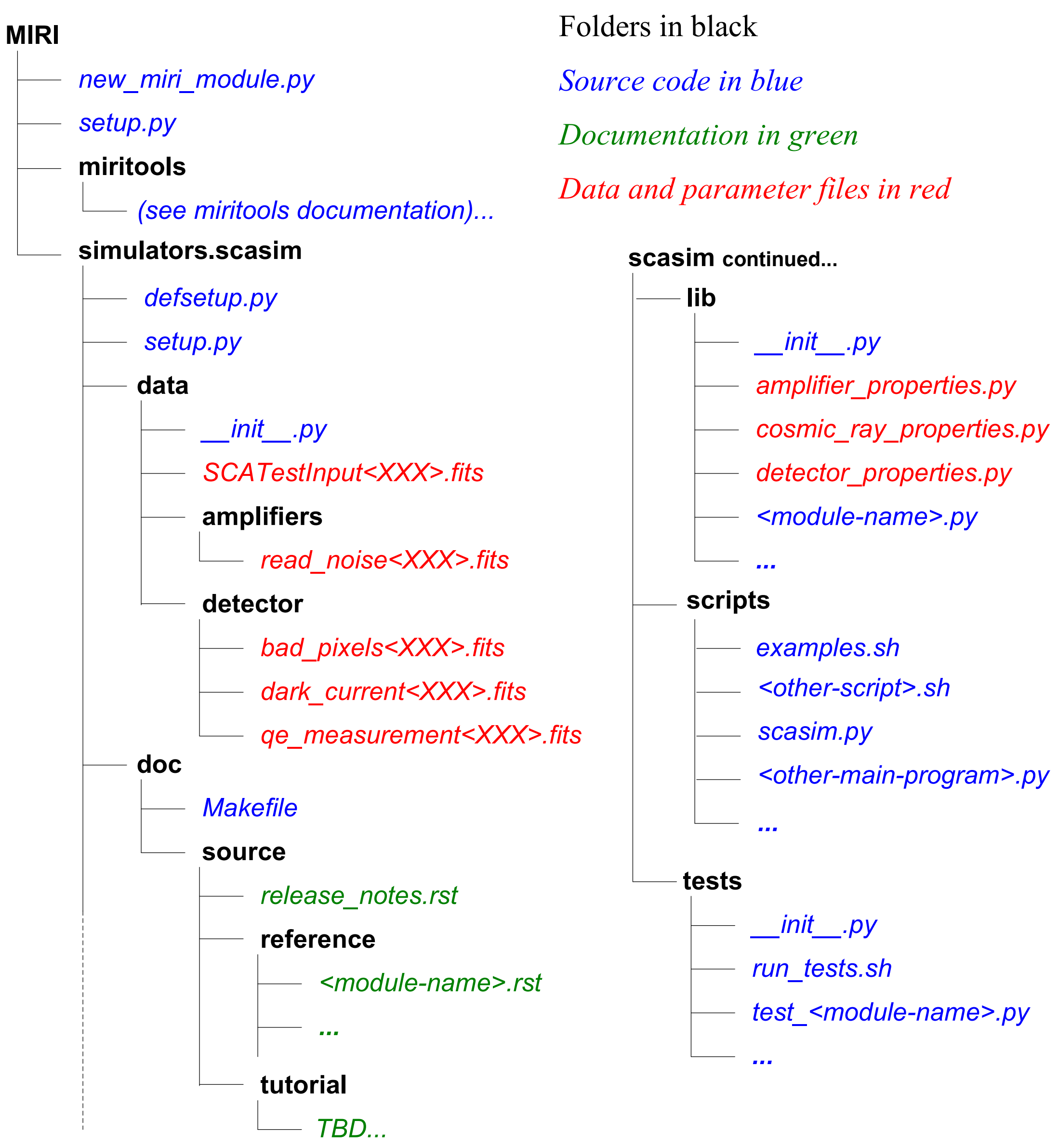

Figure 2: Overall Layout of the SCA Simulator Files

4.4 Building the SCA simulator

4.4.1 Building the software for the first time

Before building the SCA simulator for the first time, first decide where the modules will be installed. By default, the build script will install them into the same directory where other Python utilities are installed. If you don't have permission to write to those directories, the "–`prefix`" directive can be used to tell the build script to install the modules somewhere else. Before building the SCA simulator, ensure that the PATH and PYTHONPATH environment variable includes the "bin" and "lib/…/site-packages" directories where the executable files will be installed. For example:

```
export PATH="/my/alternative/bin:${PATH}"
export PYTHONPATH="/my/alternative/lib/python2.6[1]/site-packages:${PYTHONPATH}"
```

[1] Assuming you are using Python 2.6.

if you plan to install the software in the directory `/my/alternative.` It is important that you modify your shell startup file (e.g. .tcshrc or .bashrc) so the PATH and PYTHONPATH are defined each time you log in. The MIRI software suite (including the SCA simulator) can be built with this command (executed from the “miri” or “teams/miri” directory created in section 4.2):

```
python[2] setup.py install
```

or with this command if you want the installation written to `/my/alternative` instead:

```
python[2] setup.py install –prefix=/my/alternative
```

When the build procedure has finished, log out and log back in again so that your shell can locate the new executables.

NOTE: If you have installed the MIRI Python software using MIRICLE, the command “ur_setup” should define the necessary environment variables.

4.4.2 Rebuilding the software

If you need to rebuild the software (e.g. after an update) it is a good idea first to clean the old version with the command:

```
python setup.py clean
```

and then repeat the build procedure described above. *Note that new versions of modules will only become known to the installed system after rebuilding the software*. If you run find that a new release of the software is not building properly (e.g. some old features or old configurations refuse to go away) the old installation can be completely removed with the commands:

```
rm –rf ./build
rm –rf /my/alternative/lib/python2.6/site-packages/miri
rm `find . –name “*.pyc”`
```

This will force a complete rebuild from scratch.

4.5 Testing the SCA simulator

4.5.1 Simple check

The simplest way to check that the SCA simulator has been installed correctly is to try importing it from Python. The __version__ string will tell you which version is installed and the __doc__ string will give some information.

```
python
>>> import miri.simulators.scasim
>>> print miri.simulators.scasim.__version__
>>> print miri.simulators.scasim.__doc__
```

If the import does not work, check your PATH and PYTHONPATH are defined correctly, try logging out and logging in again, and if that doesn’t work try repeating the build procedure again (and watch for any errors).

4.5.2 Unit tests

If the simple check above has worked, you can run a much more rigorous test of the scasim installation by running the unit tests. First go to the test directory:

[2] On a machine in which there are several versions of Python installed, you may need to refer to an explicit version of Python compatible with stsci_python. For example the command “python2.6” will run Python V2.6. Throughout this document, please substitute the command “python” with the one appropriate for your installation (or set up a command alias pointing to the correct version of Python).

```
cd miri/simulators/scasim/tests
```

Any of the individual tests can be run by executing them with Python. For example:

```
python test_detector.py
python test_sensor_chip_assembly.py
```

but for convenience all the tests can be run in sequence by executing the top-level "teams/miri/miri_run_tests.py" script. If these tests succeed, the scasim software is installed successfully. It is also useful to repeat the tests after making changes to ensure none of the changes have accidentally broken part of the scasim installation.

Note: Most tests take a few seconds, but the final test (running the simulator with test data with all important combinations of parameters) can take a few minutes.

4.5.3 Ad-hoc tests

The scasim software also comes with a collection of ad-hoc tests. Each library module runs an ad-hoc test when it is run as a main program. For example:

```
cd miri/scasim/lib
python detector.py
```

The main differences between the unit tests and the ad-hoc tests are:

- The unit tests are more rigorous – they will test all the important combinations of parameters. By comparison, the ad-hoc tests run a few common or interesting examples.
- The unit tests run silently – you only see short messages telling you how many of the tests have been run. By comparison, the ad-hoc tests run in verbose mode and give a running commentary.
- The unit tests do not use the plotting utilities but the ad-hoc tests do.
- The unit tests tidy up everything they create, so no files are left behind. By comparison, the ad-hoc tests will leave a collection of example files in the "data" directory (which can be inspected and tidied up manually).

4.5.4 Examples

The scasim software includes a script which contains a series of commands giving examples of how to use the software. The software can also be tested by running this script:

```
cd miri/simulators/scasim/scripts
source ./example.sh
```

This example script behaves in a similar way to an ad-hoc test. Some of the commands generate plots and run in verbose mode, and the script will leave behind a collection of files in the current working directory which will need to be tidied up manually.

4.6 Deployment Summary

Figure 3 shows a summary of the complete SCASim deployment procedure (which existed before the MIRICLE script):

1) Unless it is already installed on your system, the first step is to download and installed Python V2.5, V2.6 or V2.7 from the "python (x,y)" web site (e.g. http://www.pythonxy.com/, which includes numpy, scipy and matplotlib in one package, together with a choice of development IDEs) .

2) STScI Python utilities should then be downloaded from the STScI Python web site (http://www.stsci.edu/resources/software_hardware/stsci_python) and installed to provide all the STScI extensions (such as pyfits).

3) The "prototypes/cr_ramp_fit" and "teams/miri" modules are then downloaded from the JWST assembla repository, as described in section 4.2 and built as described in section 4.4.

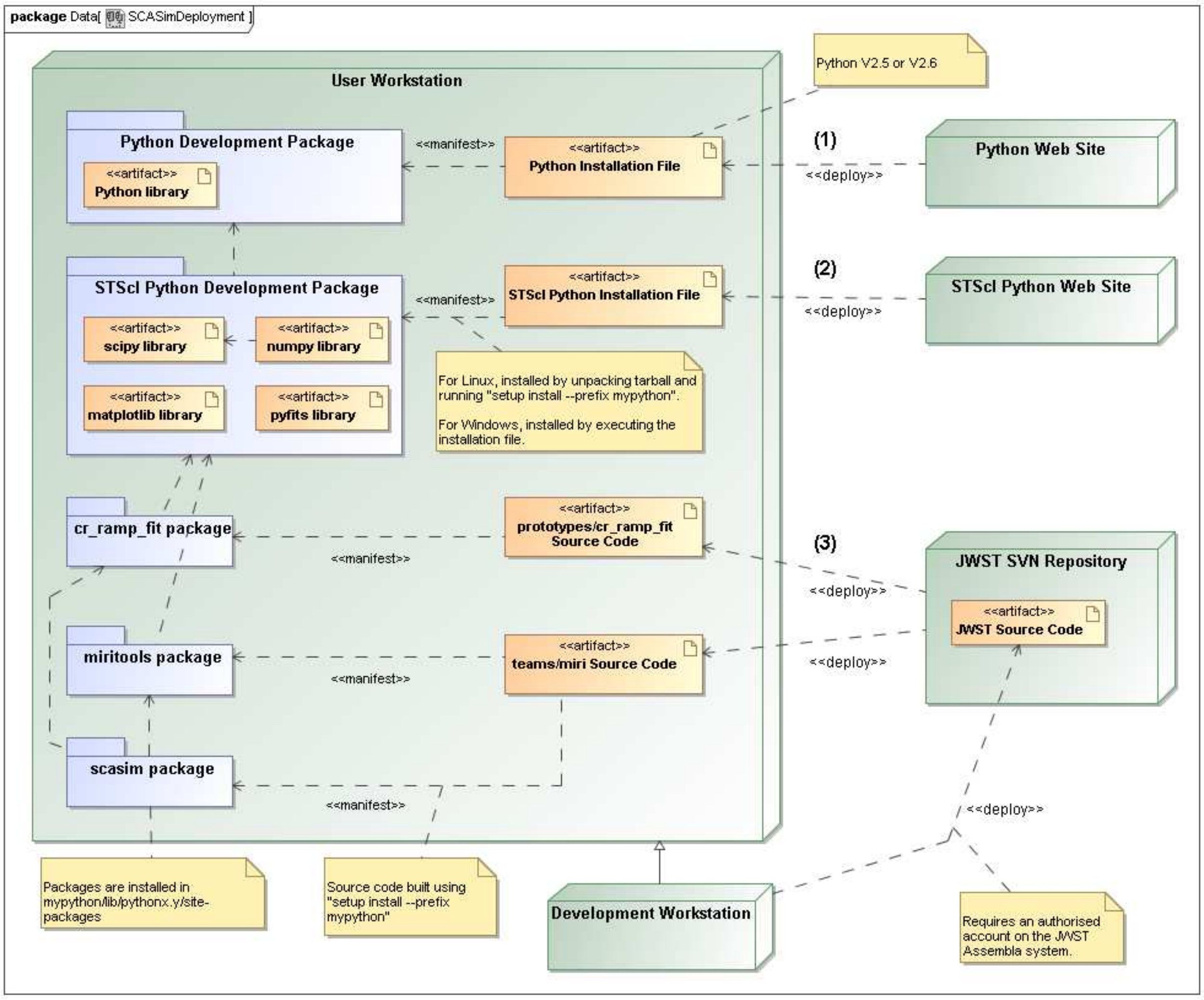
package Data[SCASimDeployment]
User Workstation
Python V2.5 or V2.6
Python Development Package
<<artifact>> Python library
<<manifest>>
<<artifact>> Python Installation File
(1)
<<deploy>>
Python Web Site
STScI Python Development Package
<<artifact>> scipy library
<<artifact>> numpy library
<<artifact>> matplotlib library
<<artifact>> pyfits library
<<artifact>> STScI Python Installation File
(2)
STScI Python Web Site
For Linux, installed by unpacking tarball and running "setup install --prefix mypython".
For Windows, installed by executing the installation file.
cr_ramp_fit package
<<artifact>> prototypes/cr_ramp_fit Source Code
(3)
JWST SVN Repository
<<artifact>> JWST Source Code
miritools package
<<artifact>> teams/miri Source Code
scasim package
Packages are installed in mypython/lib/pythonx.y/site-packages
Source code built using "setup install --prefix mypython"
Development Workstation
Requires an authorised account on the JWST Assembla system.

Figure 3: SCASim Software Deployment Summary

5 OBTAINING INPUT DATA

5.1 Input file format

The SCA simulator is designed to read detector illumination files created by other MIRI instrument simulators. The file format is described in the software design document, [1], and summarised in Figure 4. The file is in FITS format, consisting of a primary header HDU followed by extension HDUs containing intensity data, wavelength data and direction data. Only the intensity data is compulsory, and the direction data is not used by the current version of the simulator.

The primary FITS header must contain all the standard header parameters that need to be forwarded into the level 1 FITS file (as described in [14]). In particular, the SCA simulator will recognise the following keywords in the header:

Keyword	Description
SCA_ID	An integer keyword used to identify the SCA module being simulated (e.g. 493, 494, 495).
DETECTOR	A string describing the SCA module being simulated (e.g. “MIRIMAGE”, “MIRIFULONG” or “MIRIFUSHORT”. This keyword is used only if the SCA_ID keyword is absent.

SUBARRAY	A string keyword indicating whether the file contains full frame or subarray data. There are two possibilities: 1) If the keyword is absent the data will be assumed to be full frame. 2) The keyword contains a string naming the standard subarray mode that has been simulated (see page 19).

The intensity data describes the intensity of the light arriving at the detectors in photons per second. The simplest form of intensity data is a 2-D array of size (columns x rows) giving the photon flux arriving at each detector pixel. If the wavelength data is absent the simulator will integrate on this flux and will ignore the quantum efficiency (which needs wavelength information). The wavelength data may be provided in a number of different ways. The first example is shown in Figure 5 where the 2-D intensity array is supplemented by a 2-D wavelength array giving the wavelength of the light arriving at each detector pixel. The figure is simplified by showing only 4 wavelength steps. This format would be used by a MIRI spectrograph, where the wavelength of the incident light varies across the detector surface. The simulator will take into account the quantum efficiency when integrating on this flux.

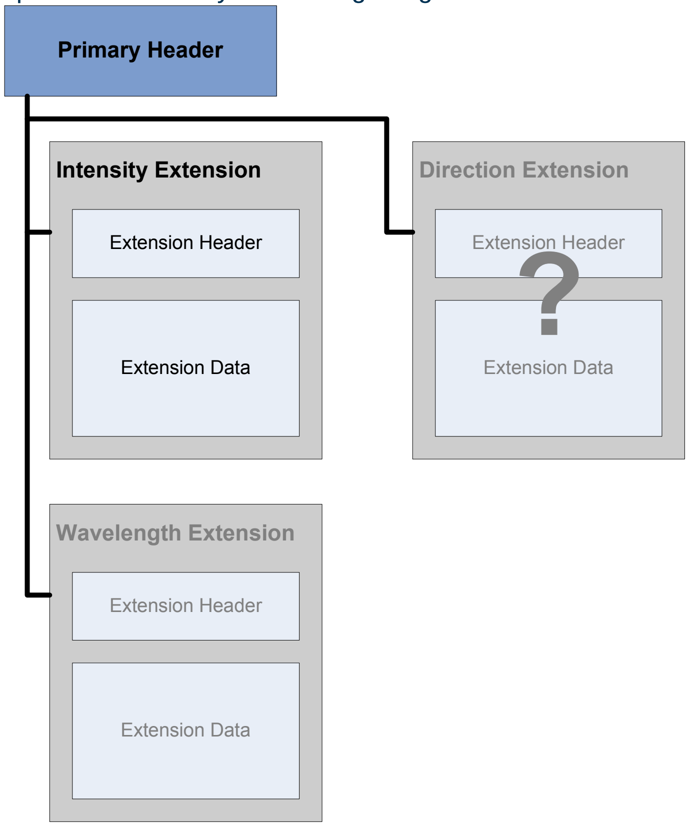

Figure 4: Detector Illumination FITS File Format

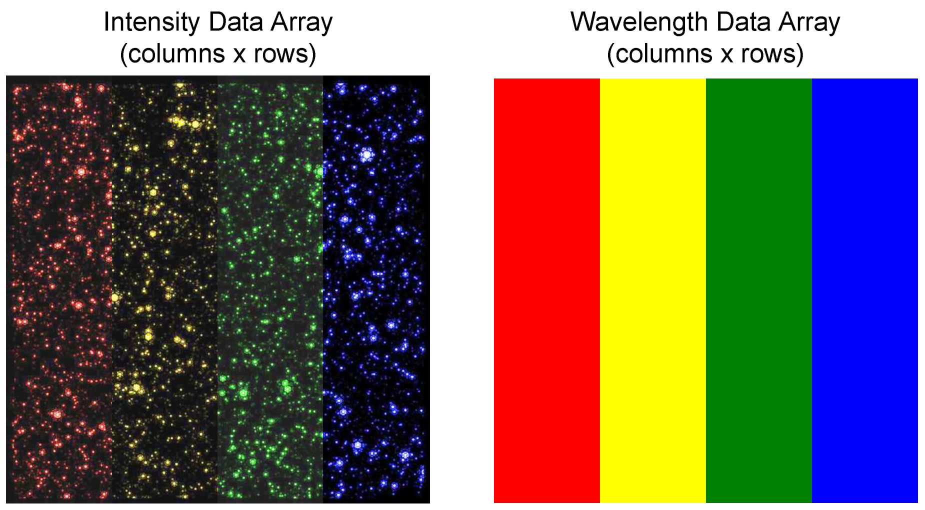

Figure 5: Example Detector Illumination Data for a MIRI Spectrograph

Another example is shown in Figure 6, where the intensity array is provided as a 3-D array containing a series of slices. 4 slices are shown, but there can be any number. Each of these slices will be matched to a corresponding slice in the wavelength array. In the case of Figure 6, each intensity slice describes light of just one wavelength, so the wavelength array can be of size (1 x 1 x slices) instead of (columns x rows x slices). (Both arrangements would work, but the SCA simulator assumes that if a single wavelength value is provided per slice it broadcasts to all columns and rows, and the first arrangement gives a smaller file size.) A multi-slice intensity array would typically be written by an imager simulator, where the intensity and wavelength along the slices follow the profile of the filter used by the imager.

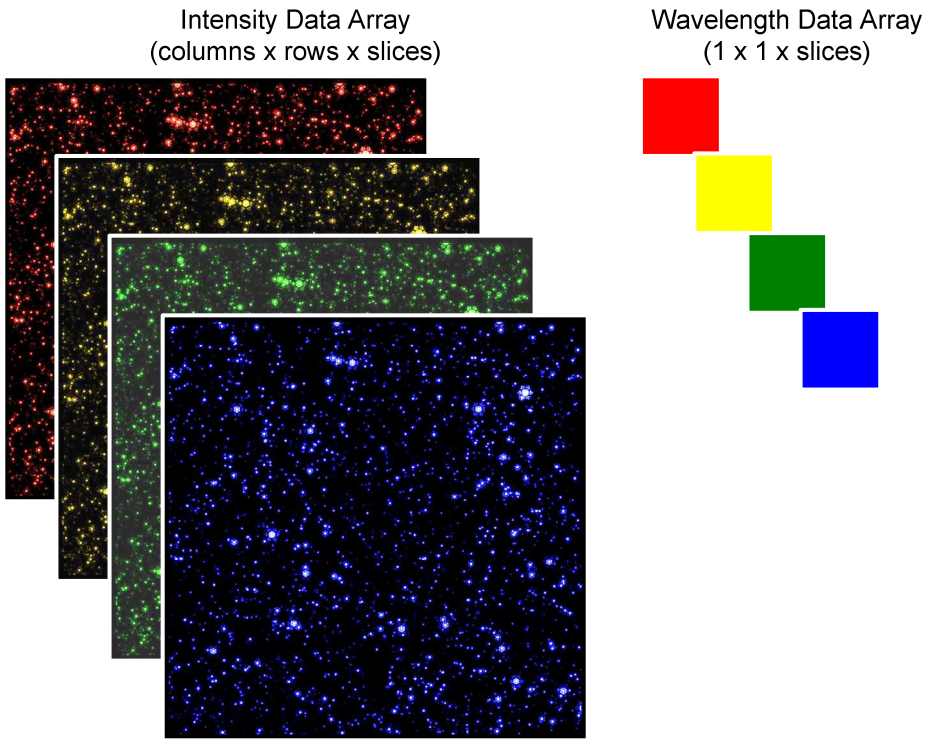

Figure 6: Example Detector Illumination Data for a MIRI Imager

It is possible to build up more complicated examples than the ones used here — for example there could be many slices like Figure 5 to describe multiple light sources each with a different variation in wavelength across the detector (as might happen with slit-less spectroscopy). The general rule is that the simulator will combine together whatever intensity and wavelength slices it finds in the file. Having said that, the quantum efficiency curve tends to be fairly flat at central wavelengths and small variations in the wavelength don't make much different to the simulation, so it isn't worth constructing elaborately complicated detector illumination files.

5.2 Generating test data from other formats

If you would like to try the SCA simulator but don't have any detector illumination files written by other MIRI simulators, it is possible to make your own test data by using the command:

```
make_sca_file.py <inputfile> <outputfile>
     --filetype <> --scale <> --wavmax <> --wavmin <>
```

The command reads a specified input file (containing the intensity data) and converts it to an output file in the detector illumination format. The optional parameter `--filetype` gives the type of the input file, which can be ASCII (where the intensity values are written as lines in an ascii file) or IMAGE (where the intensity values are read from a standard JPEG or PNG image). ASCII format can be tedious for all but the smallest sets of data, but IMAGE format can be used to convert any interesting astronomical image into test data. All the intensity values are multipled by the `--scale` parameter if one is provided.

Although the input file contains only intensity data, wavelength data can be provided by specifying the `--wavmax` and `--wavmin` parameters. These will create a wavelength ramp looking like Figure 5 ranging from the minimum and maximum specified.

The "make_sca_file" utility is described in more detail in the SCASim software reference manual, [2].

5.3 Generating calibration data

The "make_sca_calibration" command is available for generating artificial calibration data:

```
make_sca_calibration.py <rows> <columns> <outputfile>
     --pattern <> --constant <> --rowslope <> --colslope <>
     --wavmax <> --wavmin <>
```

The utility generates a detector illumination file containing intensity data of size (columns x rows) to a specified pattern. The available patterns are:

- **CONSTANT**: The intensity data is filled with the value specified in `--constant.`
- **SLOPE**: The intensity data is a flat plane which has a constant offset of `--constant` and varies by `--rowslope` along the rows and `--colslope` along the columns.
- **BADPIXEL**: A bad pixel map is generated containing `--constant` randomly placed bad pixels.

Wavelength data can be provided with the CONSTANT or SLOPE data by specifying the `--wavmax` and `--wavmin` parameters, as with "make_sca_file". The CONSTANT pattern can be used to create artificial perfect dark or flat-field frames (with `--constant` set to 0.0 or 1.0 respectively).

This utility is also described in more detail in the SCASim software reference manual, [2].

6 RUNNING THE SIMULATOR

6.1 The scasim command

The SCA simulator is run from the following command. See the SCASim software reference manual, [2], for a full description of the command options and parameters and see the MIRI Operational Concepts Document, [5], for a description of the readout modes and subarray modes described here.

```
scasim.py <inputfile> <outputfile>
      --detector <> --fringemap –rdmode <>  --subarray <> --noburstmode
      --inttime <> --ngroups <> --nints <> --wait <>
      --temperature <>  --crmode <>
      --format <> --datashape <> --scale <>
      --cdprelease <> --previousfile <>
```

The simulator reads detector illumination data from the specified input file and writes simulated level 1 FITS data to the specified output file. The optional command parameters are:

- **`--detector`**: The identity of the detector module to be simulated ('MIRIMAGE', 'MIRIFULONG' and 'MIRIFUSHORT' – see [11]). If not specified, the input file will be examined and the value specified in the DETECTOR keyword in the primary FITS header used. Failing that, the default detector is 'MIRIMAGE'.

- **`--fringemap`**: The path+name of a FITS file containing a fringe map to be used to modulate the detector illumination. If not specified, no fringe map will be applied.

- **`--rdmode`**: The detector readout mode ('SLOW', 'FAST', 'FASTGRPAVG 'or 'FASTINTAVG' – see [5]). If not specified, 'FAST' will be used.

- **`--subarray`**: The output subarray mode, if needed ('FULL', 'MASK1550', 'MASK1140', 'MASK1065', 'MASKLYOT', 'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64', 'SLITNESSPRISM or ''FASTINTAVG' – see [5]). If not specified 'FULL' will be assumed. (The input subarray mode will be inferred from the value of the SUBMODE keyword in the primary FITS header of the input file.)

- **`--noburstmode:`** Include this parameter if subarray data are not to be simulated in burst mode.

- **`--inttime`**: The integration time in seconds. If not specified, the integration time will be derived from the read mode and `--ngroups`. Use shorter times when the test data is small and when the detector readout mode is FAST.
  NOTE: Any requested integration time will be rounded up to the time resulting from the nearest whole number of groups.
  NOTE: It is not sensible to specify both `--inttime` and `--ngroups`. If both parameters are specified, `--ngroups` will set the integration time and the `--inttime` value will be ignored. If neither parameter is specified, the default values associated with the detector readout mode are used.

- **`--ngroups`**: The number of groups making up each integration. If not specified `--ngroups` will be derived from the detector readout mode and integration time. If both parameters are specified, `--ngroups` takes priority over `--inttime`.

- **`--nints`**: The number of integrations. The total exposure time will be `inttime` x `nints`. If not specified, the default number of integrations associated with the detector readout mode will be used.

- **`--wait:`** The time that elapses in between exposures.

- **`--temperature`**: The detector temperature in K. If not specified it will default to the expected target temperature contained in the detector properties file.

- **`--crmode`**: The cosmic ray environment ('NONE', 'SOLAR_MIN', 'SOLAR_MAX' or 'SOLAR_FLARE'). If not specified, 'SOLAR_MIN' will be used.
  NOTE: 'SOLAR_FLARE' simulations can take a long time because of the high number of cosmic ray events involved. Use very short exposures.

- **`--format`**: The file format required ('FITSWriter' or 'STScI'). FITSWriter is the file format written by the FM software and recognised by DHAS, [7], and 'STScI is the level 1b format recognised by the JWST DMS pipeline software, [6]. If not specified, the default format is 'STScI'.

- **`--datashape`**: The data array shape required ('cube', 'hypercube' or 'slope'). This controls whether each integration is saved to its own separate data cube ('hypercube') or if all integrations are concatenated into one large data cube ('cube'). The 'slope' option applies a straight line fit to the data to generate one 2-D image per integration. If not specified, the default format is 'hypercube' for 'STScI' format and 'cube' for 'FITSWriter' format.

- **`--scale`**: A scale factor with which to multiply all intensity data as it is imported into SCAsim. This factor is normally fixed at 1.0. It can be altered on rare occasions where it is necessary to bring faulty input data into the flux range expected by SCAsim.

- --cdprelease: The CDP release from which CDP files are to be imported. Do not change this parameter unless you know which CDP files will be imported. Defaults to the latest release.

- **`--previousfile`**: The name of another input file, in SCA format, which describes the detector illumination for the previous exposure. The contents of this input file are processed first and leave behind persistence effects.

The following options can be used to vary the way in which the simulator runs:

- **`--silent`**: Run silently.

- **`--verbose`**: Run in verbose mode (with extra output).

- **`--debug`**: Run in debugging mode (with even more output).

- **`--plot`**: Generate data plots.

- **`--clobber`**: Overwrite the output file if it already exists.

in addition, these flags can be used to run a simulation with particular effects turned on or off.

- **`--noqe`:** Turn off quantum efficiency simulation.

- **`--nopoisson`:** Turn off Poission noise simulation.

- **`--noreadnoise`**: Turn off read noise simulation.

- **`--norefpixels`**: Turn off reference pixel simulation.

- **`--nobadpixels`**: Turn off bad pixel simulation.

- **`--nodark`**: Turn off dark current simulation.

- **`--noflat`**: Turn off flat-field simulation.

- **`--nogain`**: Turn off amplifier bias and gain simulation.

- **`--nolinearity`**: Turn off detector non-linearity simulation.

6.2 Some examples

Some example commands, which use the 80x64 test data included with the release.

The detector readout mode is SLOW and the exposure will be made of 12 groups and 2 integrations. (It is better to give the number of groups explicitly because the 80x60 data has a very small frame time and can easily convert a small integration time into hundreds of groups). The detector temperature is 7.0K and the simulation will assume a cosmic ray environment of SOLAR_MAX. Level 1 FITS data will be saved to SCATestOutput1.fits:

```
scasim.py data/SCATestInput80x64.fits data/SCATestOutput.fits --rdmode SLOW
      --ngroups 12 --nints 2 --temperature 7.0 --crmode SOLAR_MAX
```

As above but with FASTGRPAVG readout mode and overwrite the output file. The resulting file will contain 3 sets of (4) averaged groups:

```
scasim.py --clobber data/SCATestInput80x64.fits /data/SCATestOutput.fits
      --rdmode FASTGRPAVG --ngroups 12 --nints 2 --temperature 7.0
      --crmode SOLAR_MAX
```

As above but with FASTINTAVG readout mode. The 2 integrations specified will be rounded up to 4 and the file will contain 1 set of (4) averaged integrations.

```
scasim.py --clobber data/SCATestInput80x64.fits data/SCATestOutput.fits
      --rdmode FASTINTAVG --ngroups 12 --nints 2 --temperature 7.0
      --crmode SOLAR_MAX
```

In this simulation the cosmic ray environment is SOLAR_MIN and the detector temperature is raised to 8.0K (which will cause a visible increase in the dark current and readout noise). This time there are 10 groups and 1 integration.

```
scasim.py --clobber data/SCATestInput80x64.fits data/SCATestOutput.fits
      --rdmode SLOW --ngroups 10 --nints 1 --temperature 8.0
      --crmode SOLAR_MIN
```

In this example a small integration time is specified and scasim will calculate the number of groups from this integration time (caution: the number of groups can grow rapidly with integration time).

```
scasim.py --clobber data/SCATestInput80x64.fits data/SCATestOutput.fits
      --rdmode SLOW --inttime 0.1 --nints 1 --temperature 7.0
      --crmode SOLAR_MAX
```

Create some full frame test calibration data using make_sca_calibration.py and then simulate a 120 second exposure in MASK1550 subarray mode.

```
make_sca_calibration.py 1024 1024 data/SCATestInput1024x1024.fits
      --pattern CONSTANT --constant 1.0

scasim.py data/SCATestInput1024x1024.fits data/SCATestOutput_MASK1550.fits
    --rdmode SLOW --subarray MASK1550 --ngroups 10 --nints 1
    --temperature 8.0 --crmode NONE
```

6.3 Detailed simulation control

The command parameters described in section 6.1 are used to define simulation parameters which are expected to change from observation to observation. More detailed control of the simulation can be achieved (if required) by editing the configuration files supplied with the simulator. These configuration files may be found in the scasim/lib directory and are:

- cosmic_ray_properties.py: Describes the properties of the cosmic ray environment.
- detector_properties.py: Describes the properties of the detectors.
- ~~amplifier_properties.py~~[3]~~: Describes the properties of the detector readout amplifiers and electronics.~~

For a detailed description of the contents of these files, see section 8, “Defining SCA Properties” on page 28.

The best way to modify these files is to copy them to the current working directory before editing them. SCASim will search for configuration files in the following places:

[3] The amplifier properties file is no longer used because the simulation of individual amplifiers has been removed.

1. Within the current working directory.
2. Within the installed scasim directory.
3. Within the PYTHONPATH search path.

So files found within the current directory will override those present in other directories. The same is true for the configuration data files, such as bad pixel masks. SCASim will list the full path and names of all the files it has found. Configuration files are now interpreted on the fly, so there is no need to rebuild the SCASim source code after editing a file within the current working directory. If the edited files are deleted or renamed, SCASim will resort to using its original files again.

N.B. If a syntax error is introduced by editing one of these files, this will be reported when SCASim is run. Make sure all opening quotes and brackets are matched with closing ones.

7 SIMULATION DETAILS

Figure 7 shows the processing which takes place within the SCA simulator.

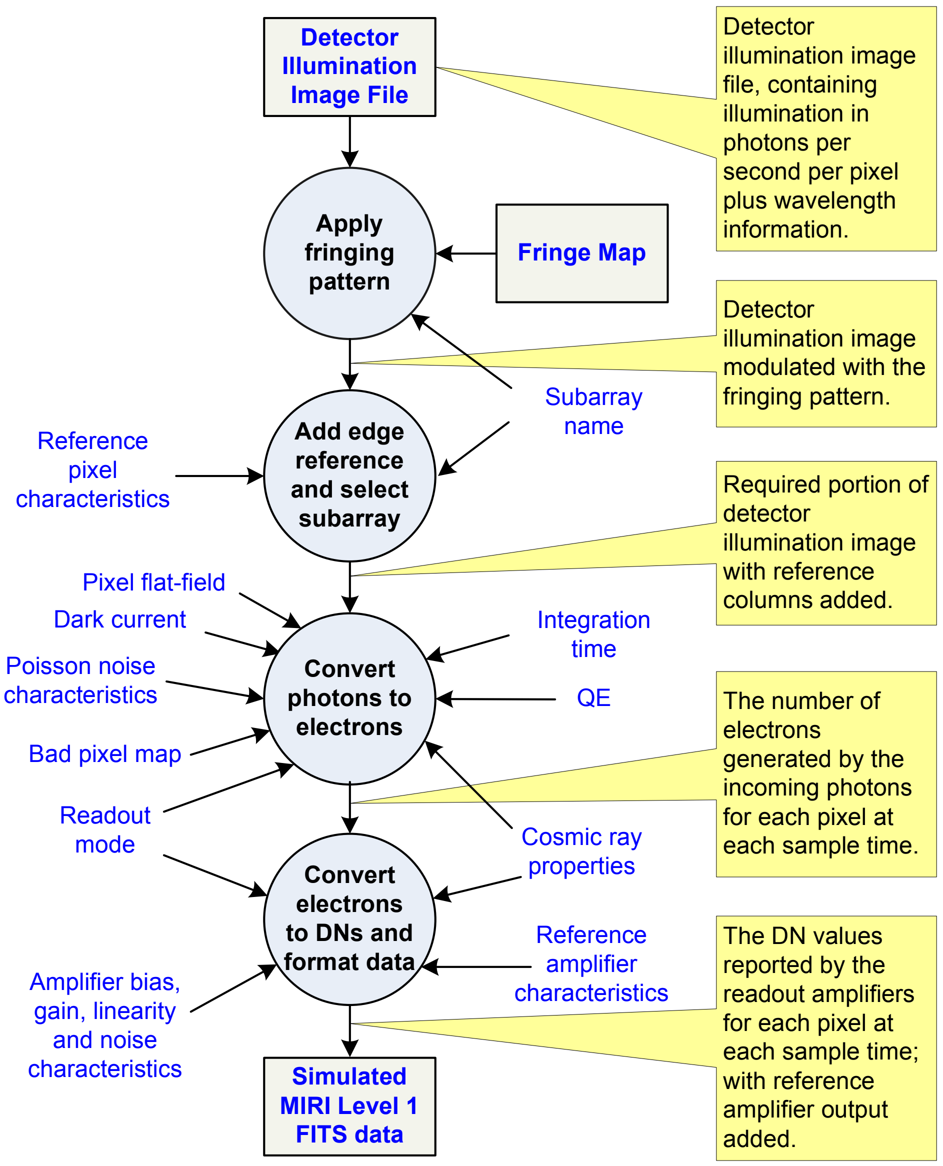
Detector Illumination Image File
Detector illumination image file, containing illumination in photons per second per pixel plus wavelength information.
Apply fringing pattern
Fringe Map
Detector illumination image modulated with the fringing pattern.
Subarray name
Reference pixel characteristics
Add edge reference and select subarray
Required portion of detector illumination image with reference columns added.
Pixel flat-field
Dark current
Poisson noise characteristics
Bad pixel map
Readout mode
Convert photons to electrons
Integration time
QE
Cosmic ray properties
The number of electrons generated by the incoming photons for each pixel at each sample time.
Convert electrons to DNs and format data
Reference amplifier characteristics
Amplifier bias, gain, linearity and noise characteristics
The DN values reported by the readout amplifiers for each pixel at each sample time; with reference amplifier output added.
Simulated MIRI Level 1 FITS data

Figure 7: SCA Simulator Data Flow Diagram

7.1 Fringing

If desired, the SCA simulator can apply a fringing pattern to the detector illumination data. The fringe map is read from an image stored in a FITS file; the software will extract the image from the first HDU encountered containing data. Once read in, the fringe map is scaled so that any negative values are clipped at 0.0 and its mean value is 1.0. The detector illumination map is then multiplied by this fringe map.

For testing purpose, it is possible to generate a fringe map from a JPEG image using the `make_sca_file.py` command described in section 5.2 on page 12. The example image in Figure 8 was extracted from a MIRI document.

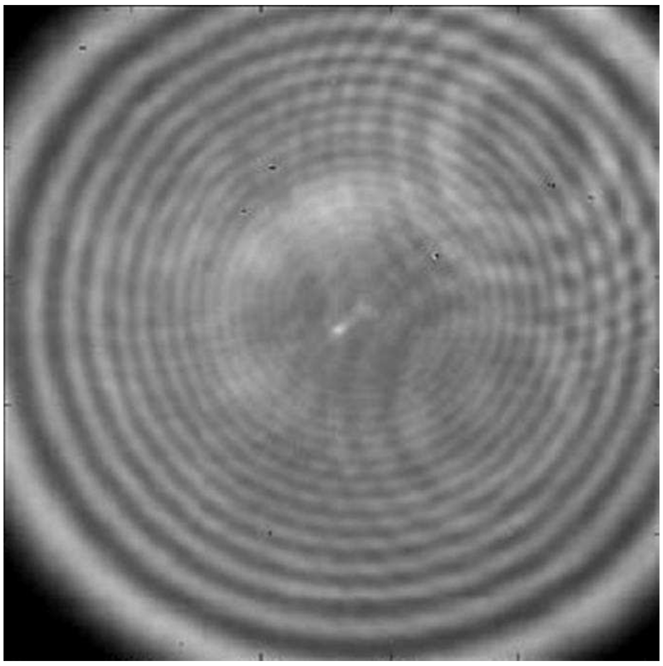

Figure 8: Example fringe pattern

7.2 Detector Integration

The SCA uses the following terminology to describe the “MULTIACCUM” readout patterns used to integrate and read the detectors:

- $\mathbf{n_{sample}}$ – the number of times the detector is sampled per readout. Using more samples reduces the readout noise at the expense of longer readout times and less time resolution. When $n_{sample} > 3$ the first and last readings in the sample are discarded. $\mathbf{n_{discarded}}$ records the number of samples discarded.

- $\mathbf{n_{frames}}$ - the number of consecutive frames combined together and averaged to generate a single group. MIRI always uses $\mathbf{n_{frames}} = 1$, so the terms “frames” and “groups” are sometimes used interchangeably for MIRI. SCASim supports both MIRI and non-MIRI modes.

- $\mathbf{n_{dropped}}$ - the number of frames ignored in between groups. MIRI always uses $\mathbf{n_{dropped}} = 0$. Some other instruments use a non-zero value, which reduces the quantity of data transmitted without reducing the integration time.

- $\mathbf{n_{groups}}$ - the number of groups of frames per integration. This is the number of readings that may be used to determine the slope for each integration.

- $\mathbf{n_{ints}}$ - the number of integrations per exposure. An integration is defined as the data collected in between resets of the detector.

The integration and readout of the detector is shown schematically in Figure 9.

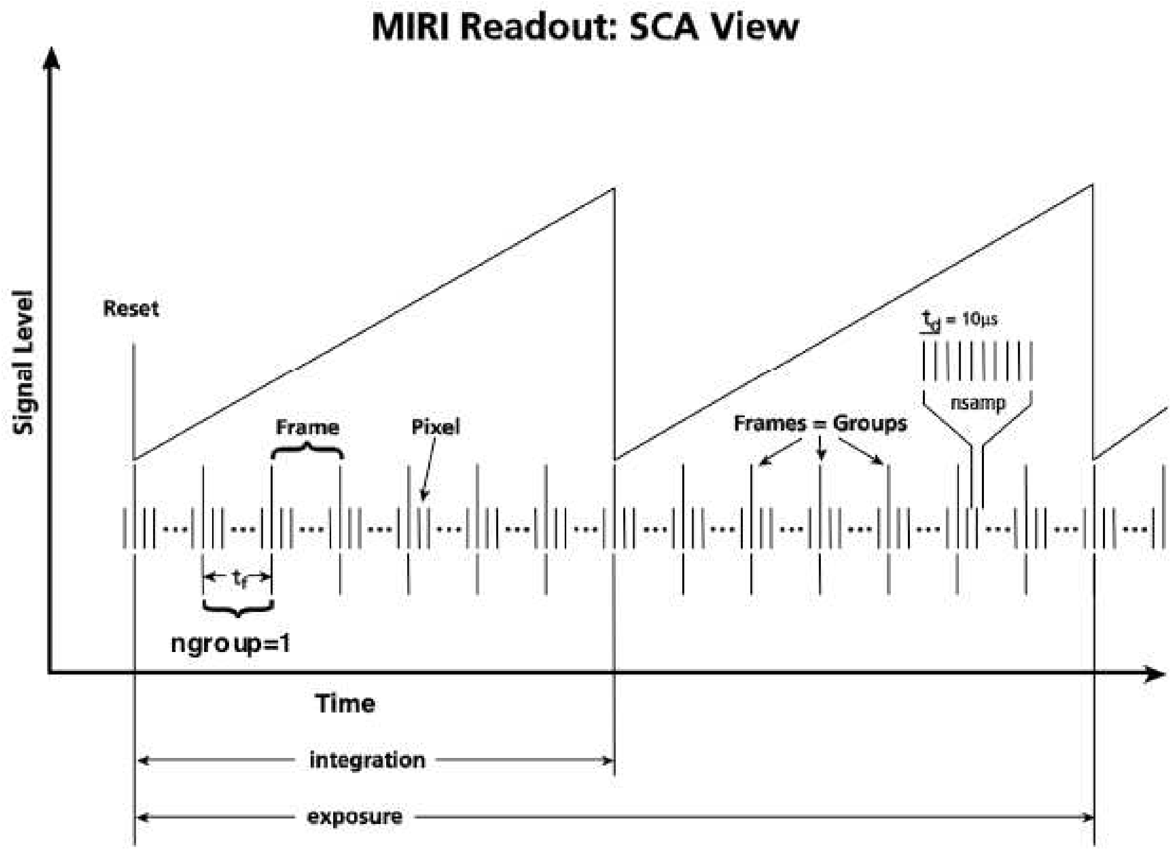

Figure 9: MIRI SCA Readout Timing Pattern

The SCA simulator offers a set of named readout modes with fixed combinations of parameters (MIRI has n_{frames} fixed at 1 and $n_{dropped}$ fixed at 0):

Name of mode	n_{sample}	n_{group}	n_{int}	Comments
SLOW	10	any	~1	n_{group} is chosen to match the observer's required integration time. n_{int} is normally 1 but can on rare occasions take other values.
FAST-short	1	1	any	n_{int} is chosen to match the observer's desired exposure time. KTC noise cannot be removed from the data.
FAST-long	1	any	1	n_{group} is chosen to match the observer's required integration time.
FASTINTAVG	1	1	any	As FAST-short, but every 4 integrations are averaged before transmission to the ground.
FASTGRPAVG	1	≥4	1	As FAST-long, but every 4 groups are averaged before transmission to the ground.

There are also two test modes:

Name of mode	n_{frames}	$n_{dropped}$	n_{sample}	n_{group}	n_{int}	Comments
SLOWGRPGAP	4	8	10	any	1	The extra frames per group and dropped groups slow the data rate and allow longer exposures to be fitted into a given output file size. For testing only.
FASTGRPGAP	4	8	1	any	1	

7.3 Subarray Modes

The SCA simulator supports the subarray modes shown in Figure 10 and the following tables:

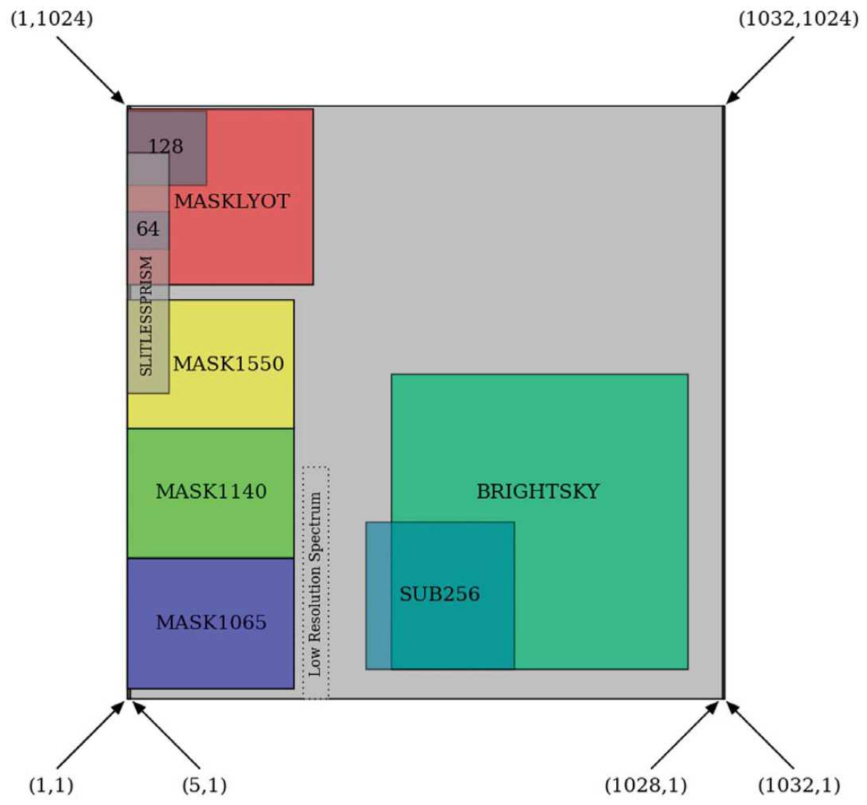

Figure 10: MIRI SCA Subarrays

Subarray name	Rows	Cols	First row	First col
FULL	1024	1032	1	1
MASK1065	224	288	19	1
MASK1140	224	288	245	1
MASK1550	224	288	467	1
MASKLYOT	304	320	717	1
BRIGHTSKY	512	512	51	457
SUB256	256	256	51	413
SUB128	128	136	889	1
SUB64	64	72	779	1
SLITLESSPRISM	416	72	529	1

7.4 Quantum Efficiency

The efficiency with which the detector converts photons into electrons is assumed to be described by the function QE(λ), where QE can vary between 0.0 and 1.0 and λ is the wavelength of the incoming photons in microns. A detector pixel could be hit by photons of many difference wavelengths during an integration (depending on the filtering and dispersing effect of the instrument). If the number of photons per second arriving at a detector pixel as a function of wavelength is represented by the function Np(λ), then the number of electrons expected to be liberated in exposure time T is given by:

$$Ne = T \int_{\lambda} Np(\lambda)\, QE(\lambda)$$

In practise, the SCA simulator receives its wavelength information in discrete chunks (see section 5.1 on page 9), so the expected number of electrons at each pixel is the sum of the electrons contributed by each chunk:

$$Ne = T \sum_{i} Np(\lambda_i)\, QE(\lambda_i)$$

The quantum efficiency function currently used by the SCA simulator is shown in Figure 11, which is taken from the measurements made during flight model testing and described in the FM "end item data pack", [11].

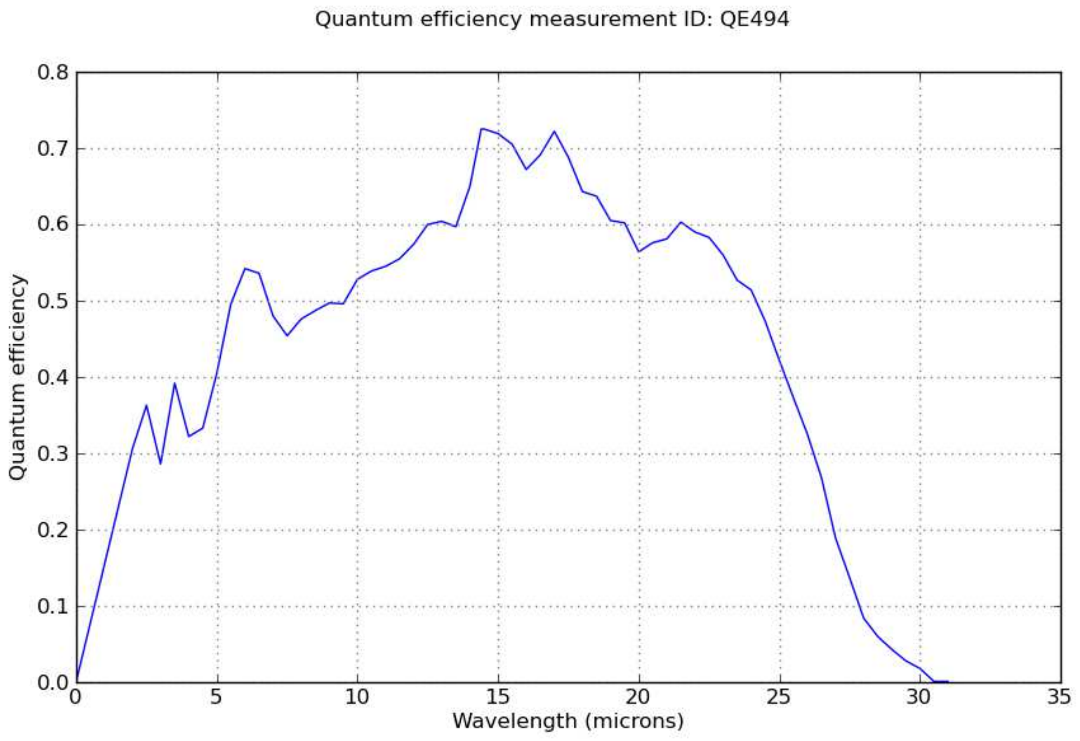

Figure 11: Quantum efficiency function

7.5 Dark Current

The detector dark current is added to the flux illuminating the detectors. If the photon flux is zero the dark current will result in the build up of a small number of electrons with time. SCAsim predicts the nominal dark current at any given detector temperature with the help of a lookup table generated from a calibration measurement. Figure 12 shows such a calibration.

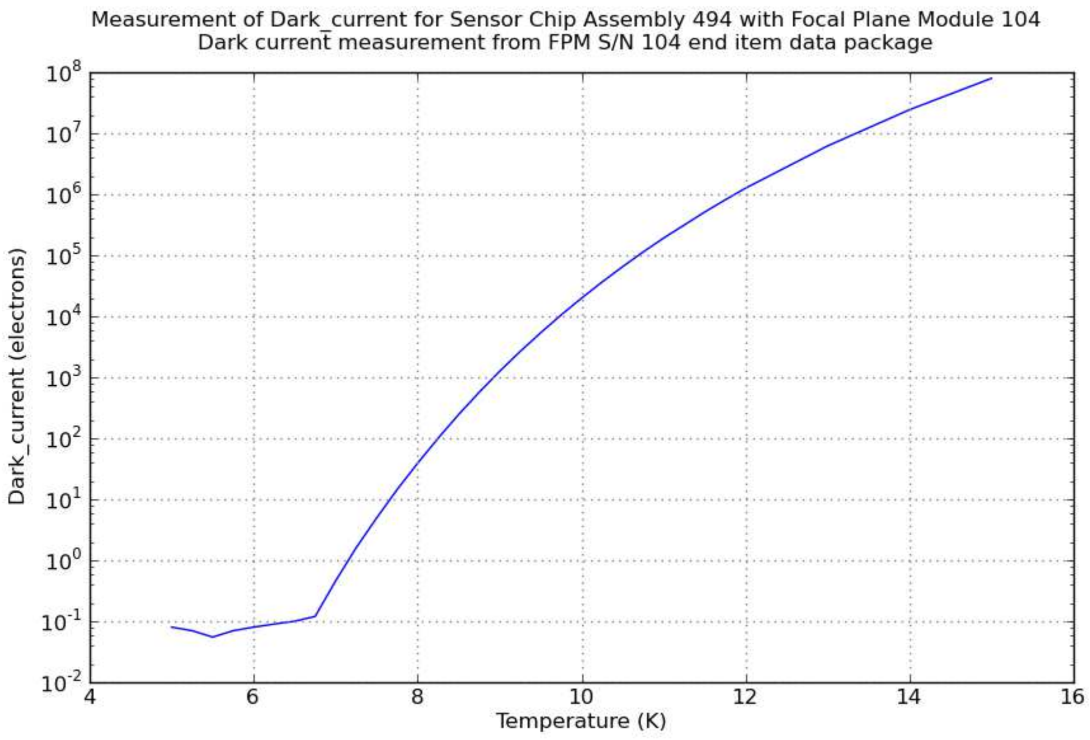

Figure 12: Dark current as a function of detector temperature

The dark current might vary from pixel to pixel over the detector surface. Deviations from the nominal dark current can be included in the simulation by given SCASim a “dark map” file, which contains multipliers to be applier to the dark current. (A file full of 1.0s will have no effect on the dark current). The dark map can also be used to identify hot pixels with an unusually large dark current.

7.6 Reference Pixels

Figure 13 shows the MIRI detector readout sequence. The detector surface has a 1024 x 1024 array of illuminated pixels plus a band of 4 columns of blind reference pixels on the left and right sides of the detector. The detector pixels are read out by 4 amplifiers which work in parallel; each one reading every 4th column, starting and ending with a reference pixel. A 5th amplifier reads out a collection of blind reference output pixels, which are ganged together. The 5 data arrays produced by the amplifiers are multiplexed together to make a data array with a “window bars” pattern, in which every 5th column comes from the reference output. In FITSWriter format, this array is then deinterlaced and the reference output pixels moved to form extra rows at the top of a reshaped array. It is this more highly compressible data array that is transmitted to the ground. This is the raw data saved in FITSWriter format. In level 1b format, the reference pixels are extracted into a separate REFOUT extension, leaving a detector frame of 1032 x1024 pixels.

The SCA simulator ignores all the intermediate steps and just simulates the final, deinterlaced array. The shape of this final array is configured by specifying these parameters:

- The number of illuminated rows and columns on the detector.
- The number of reference columns at the left hand edge of the detector.
- The number of reference columns at the right hand edge of the detector.
- The number of reference output rows at the bottom of the detector.
- The number of reference output rows at the top of the detector.

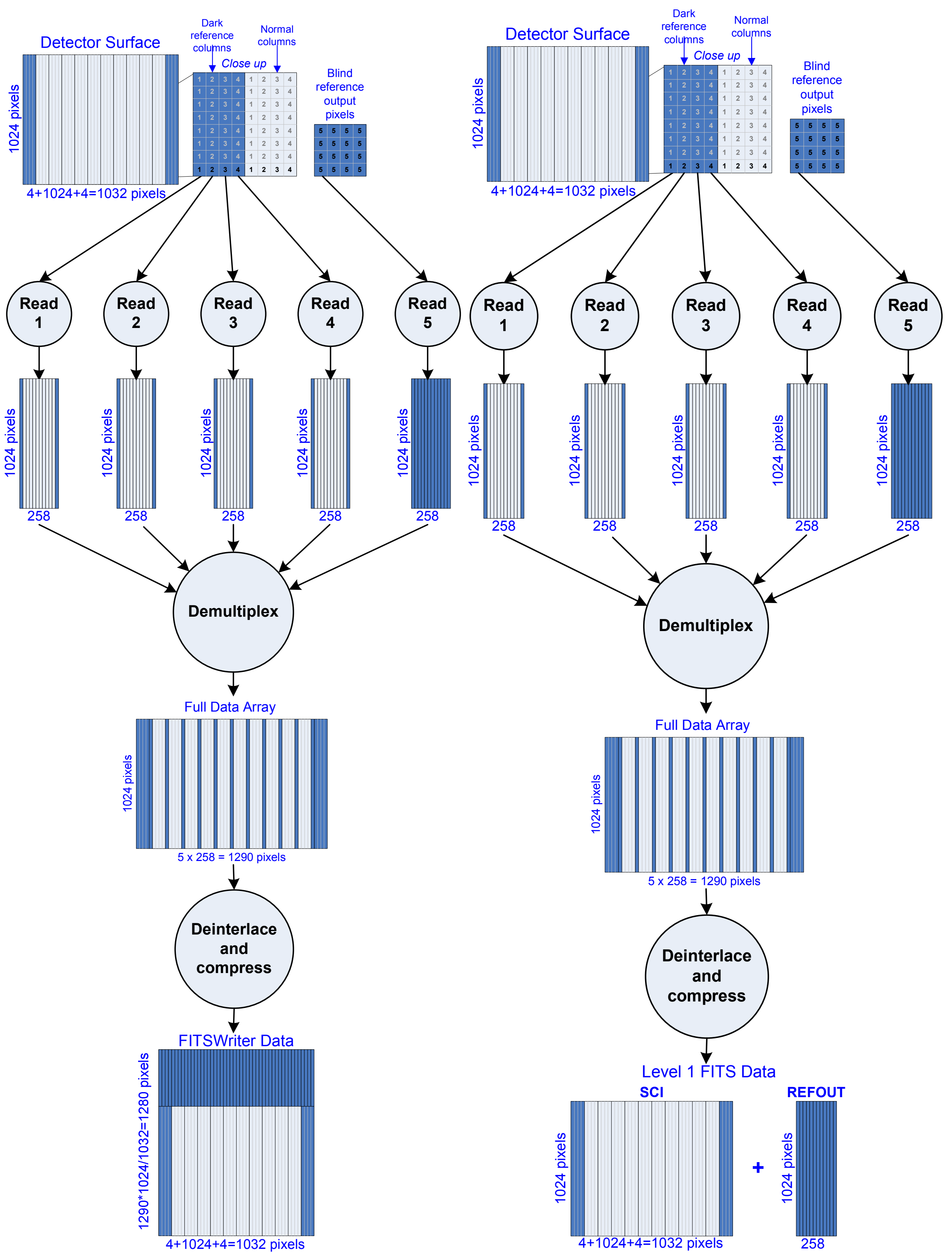

Figure 13: MIRI Detector Readout Sequence:
FITSWriter format (left) and Level 1b format (right)

The number of bottom rows is always zero for MIRI, but it is included for future flexibility. ~~In addition, the following parameters are used to configure the mapping between each amplifier and its corresponding zone on the detector surface:~~

- ~~The number of amplifiers used to read the detector, plus for each amplifier:~~

- ~~The first and last row read by that amplifier.~~
- ~~The first and last column read by that amplifier.~~
- ~~The increment between successive rows read by that amplifier.~~
- ~~The increment between successive columns read by that amplifier.~~

7.7 Poisson (Shot) Noise

The calculation made in section 7.2 gives the number of electrons that are expected to be counted in a particular pixel. However, this value is derived from the average rate of collection. If the collection rate is relatively small, the actual number counted will be determined by Poisson statistics.

The SCA simulator keeps a tab of the expected electron count as it builds up during an integration. Each time the detector is read out (non-destructively) the expected electron count Ne within each pixel is converted into an actual electron count Nc by making M random samples from a Poisson distribution of expectation value Ne. Nc is the average of all the samples obtained. The number of random samples obtained is determined by the number of times the detector is read per group, i.e.

$$M = n_{\text{frames}}\,(n_{\text{sample}} - n_{\text{discarded}})$$

where n_{frames} is the number of frames per group[4], n_{sample} is the number of samples made per frame and $n_{\text{discarded}}$ is the number of samples discarded. MIRI supports two classes of readout mode (see section 7.2 on page 17):

Readout mode	n_{frames}	n_{sample}	$n_{\text{discarded}}$	M
FAST	1	1	0	1
SLOW	1	10	2	8

So for MIRI data M will be 1 in FAST mode and 8 in SLOW mode. The electron count at each reading is constrained so that it can never be less than the count obtained at the previous reading (since, with the exception of rare electronic glitches, electrons are only added to the well during an integration).

7.8 Amplifier Characteristics

NOTE: The simulation of individual amplifiers was removed from SCASim when it was modified to import read noise and gain data from Calibration Data Products. The bias, noise and gain descriptions now apply to the detector as a whole, but the following descriptions of the amplifier effects are still valid.

The detector surface is read out by 4 amplifiers, each reading every 4th column, as shown in Figure 13. Each amplifier converts the number of electrons counted, Nc, into a voltage which is applied to an analogue to digital converter, resulting in a digital value being recorded (in data number units – DN). The way in which the amplifier converts an electron count into a DN is governed by several characteristics:

7.8.1 Bias

The amplifier might begin counting electrons from a fixed offset, or the detector itself might begin each integration with a non-zero number of electrons already present within a well. Both of these effects can be simulated by giving each amplifier a bias level in electrons.

Systematic effects in the detector or amplifier might cause the bias level to drift with each integration. This is the kind of drift the reference columns are designed to calibrate, and it is simulated in SCAsim by making small random changes to the bias level at the start of each new integration.

7.8.2 Read Noise

Each time an amplifier converts an electron count to a DN, the conversion can be affected by thermal noise in the electronics. This noise level varies with detector temperature. As with dark current, the SCA simulator looks up the read noise expected at a given detector temperature using a lookup table defined from a calibration measurement for each amplifier, such as the one shown in Figure 14.

[4] Not the number of frames per integration.

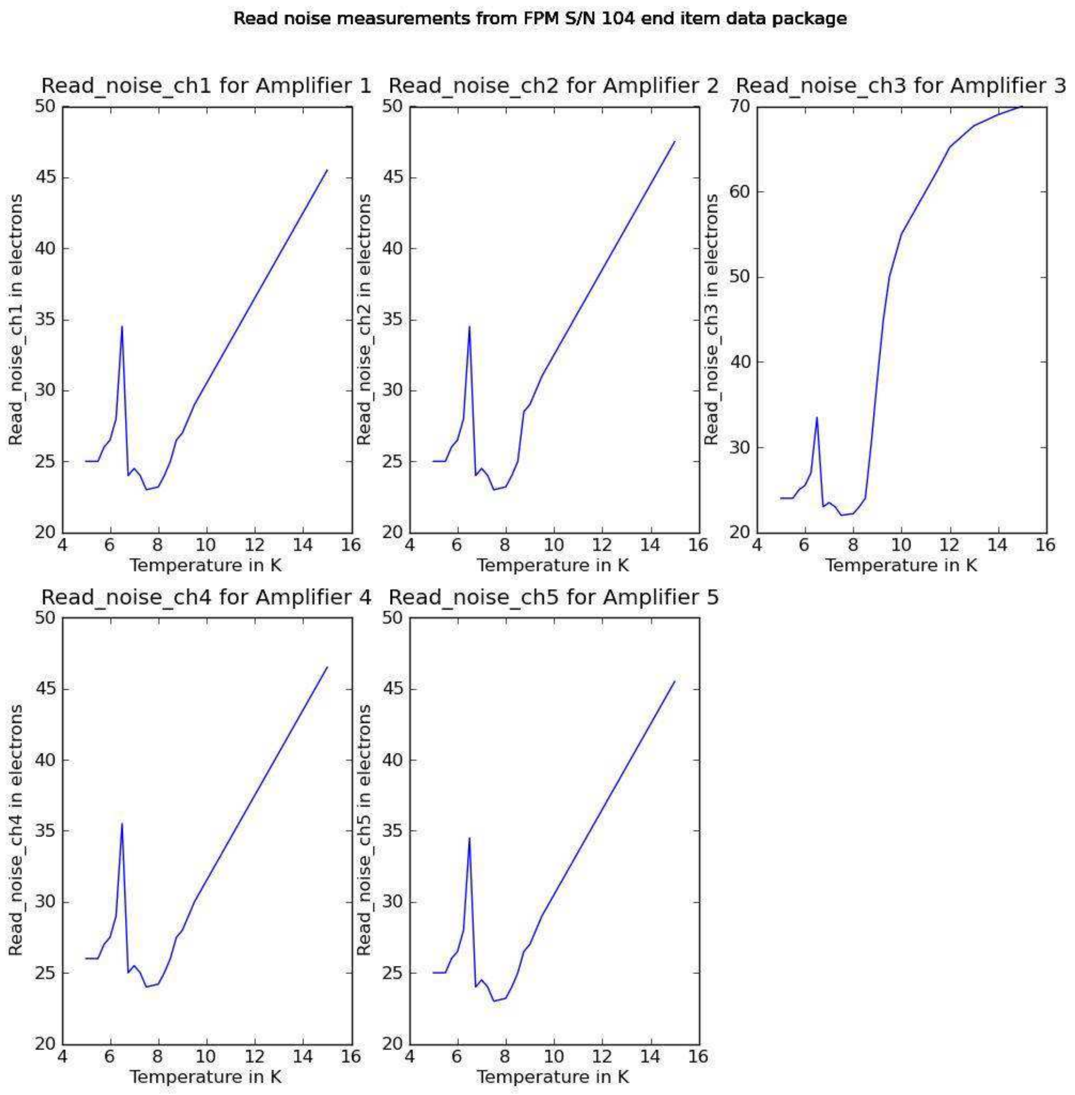

Figure 14: Read noise as a function of temperature for 5 amplifiers

The SCA simulator simulates this thermal noise by taking a random sample from a Normal distribution centred on the perfect reading whose width is determined by the read noise.

7.8.3 Gain and Linearity

The sensitivity of each amplifier, i.e. how quickly the output DN changes when the input electron count changes, is determined by the amplifier gain. The conversion function between electron count and DN also deviates from linearity at high counts. An example is shown in Figure 15.

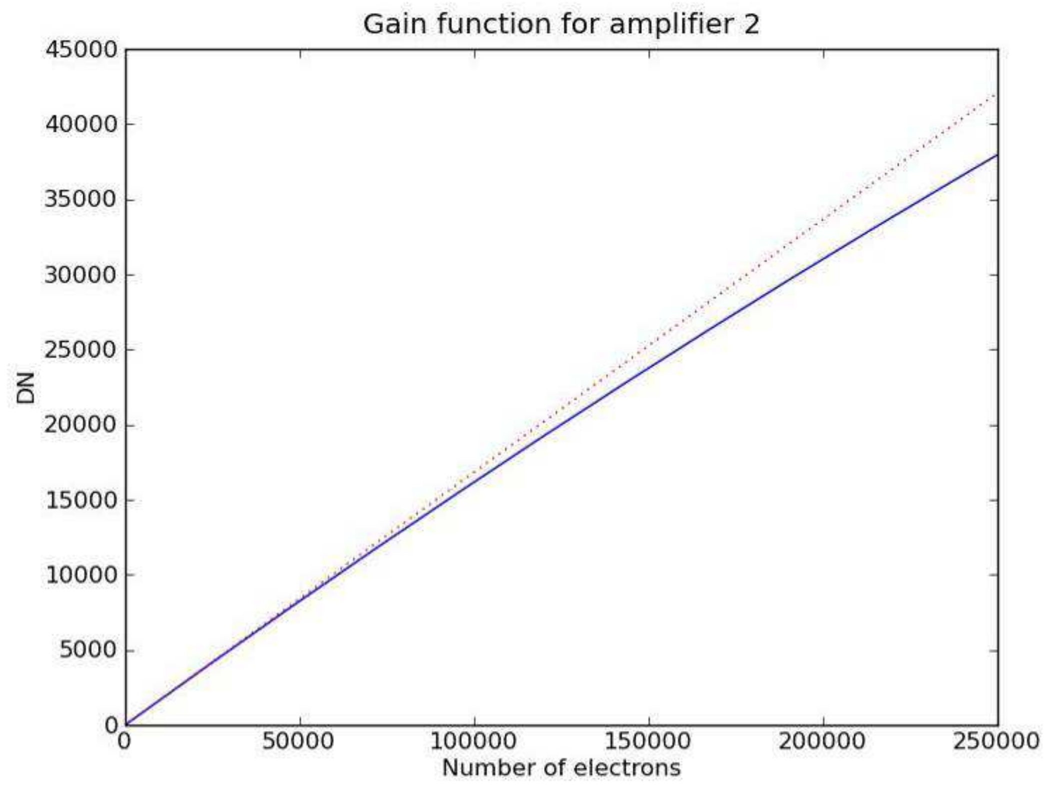

Figure 15: Data Number as a function of electron count

The gain and linearity are simulated by applying a user-defined conversion function. Two such functions are currently supported:

- **LINEAR**: A perfectly linear gain with a user-defined slope. For example, DN = slope x e.
- **POLYNOMIAL**: A polynomial function with user-defined coefficients. For example, DN = a e^2 + b e + c.

At present the gain and linearity are expected to be the same for all pixels read by a particular amplifier (although each amplifier can have a different gain, resulting in a stripy pattern in the data). Variation in gain from pixel to pixel might be added as a future upgrade.

7.9 Bad Pixels

Some detector pixels might not be functional or might have an abnormally low QE. These pixels may be identified using a “bad pixel map” containing a 0 to indicate a good pixel and a 1 to indicate a bad pixel. The SCA simulator simulates these bad pixels by giving them a QE of zero, which makes them insensitive to light.

7.10 Persistence Effects

Detector persistence effects are simulated by reducing the efficiency of the detector reset. Instead of a reset leaving zero charge in the potential wells, the residual charge is described by a transfer function, such as the one shown in Figure 16. The electron count present before reset is shown along the X axis, and the count present after a reset is shown on the Y axis. If the detector reset perfectly, the plot would be flat along the X axis. In the plot shown it can be seen that a saturated pixel containing 250000 electrons would leave behind about 13000 electrons after the first reset, then a few hundred after the second.

SCASim allows the persistence function to be defined as a polynomial by providing a list of coefficients, in a similar manner to the amplifier gain:

```
    _sca493['PERSISTENCE'] = (1.0e-8, 0.05, 0.0)   # Persistence coefficients
```

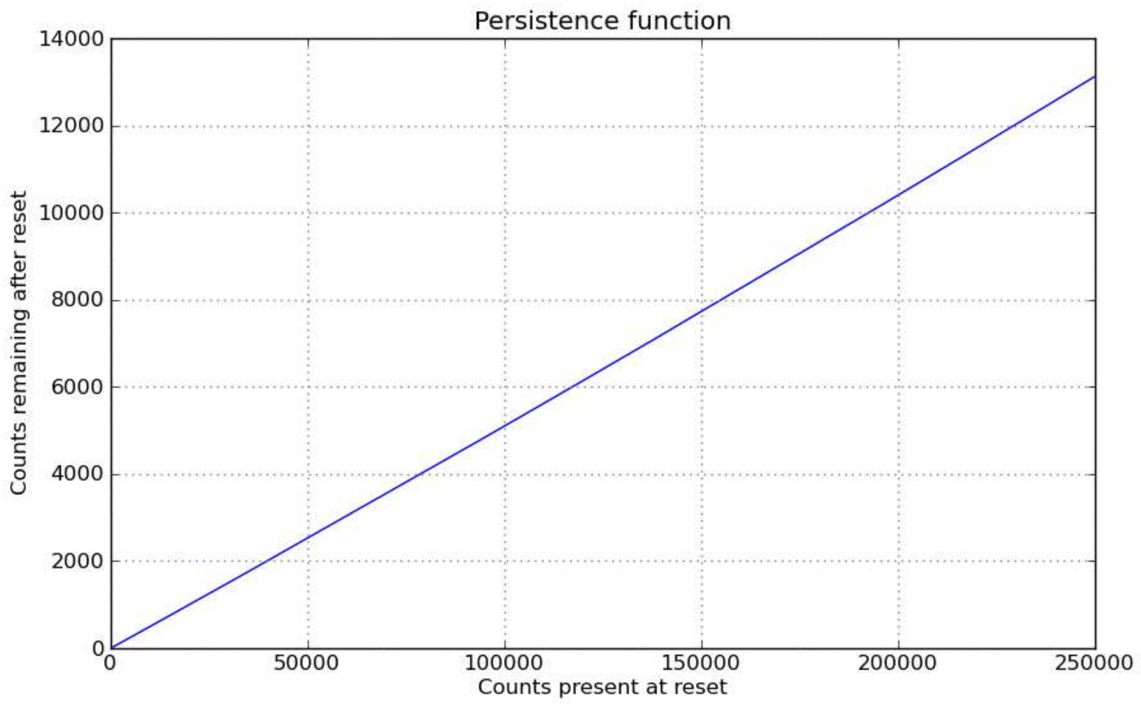

Figure 16: An example persistence function

7.11 Detector Latent Effects

See [15] for a description of the simulation of detector latent effects by SCASim.

7.12 Cosmic Ray Effects

The MIRI detectors are assumed to be targeted by cosmic rays during each integration. The incoming flux and energy distribution of the cosmic rays are governed by the cosmic ray mode, which can be:

- **NONE**: Do not simulate cosmic rays (flux assumed zero).
- **SOLAR_MIN**: During solar minimum. Ironically, the total cosmic ray flux is greater during solar minimum because the reduced intensity of the solar magnetic field allows more Galactic cosmic rays to enter the Earth environment.
- **SOLAR_MAX**: During solar maximum. Most of the cosmic rays here will be from solar wind particles. The flux of Galactic cosmic rays is reduced.
- **SOLAR_FLARE**: During a solar flare. The cosmic ray flux is so high that most MIRI observations are spoiled. It may take a long time to simulate the thousands of cosmic ray hits.

The cosmic ray flux associated with each mode is taken from [13]. Figure 17, below, shows the energy distribution contained in one of the library files. The combination of the flux, the area of the detector surface and the integration time gives the expected number of cosmic ray hits somewhere on the detector during an integration. An actual number of hits is derived from the expected number of hits by applying Poisson statistics. Cosmic ray events are then randomly chosen from the STScI cosmic ray library, [13], and are targetted at random coordinates over the detector surface. The energy signature described in the library is then applied to the detector pixels surrounding those coordinates. 99% of the time the cosmic ray will cause the electron count to jump upwards, but every now and then a downwards jump will be simulated, as described in [12]. See section 8.2.4.

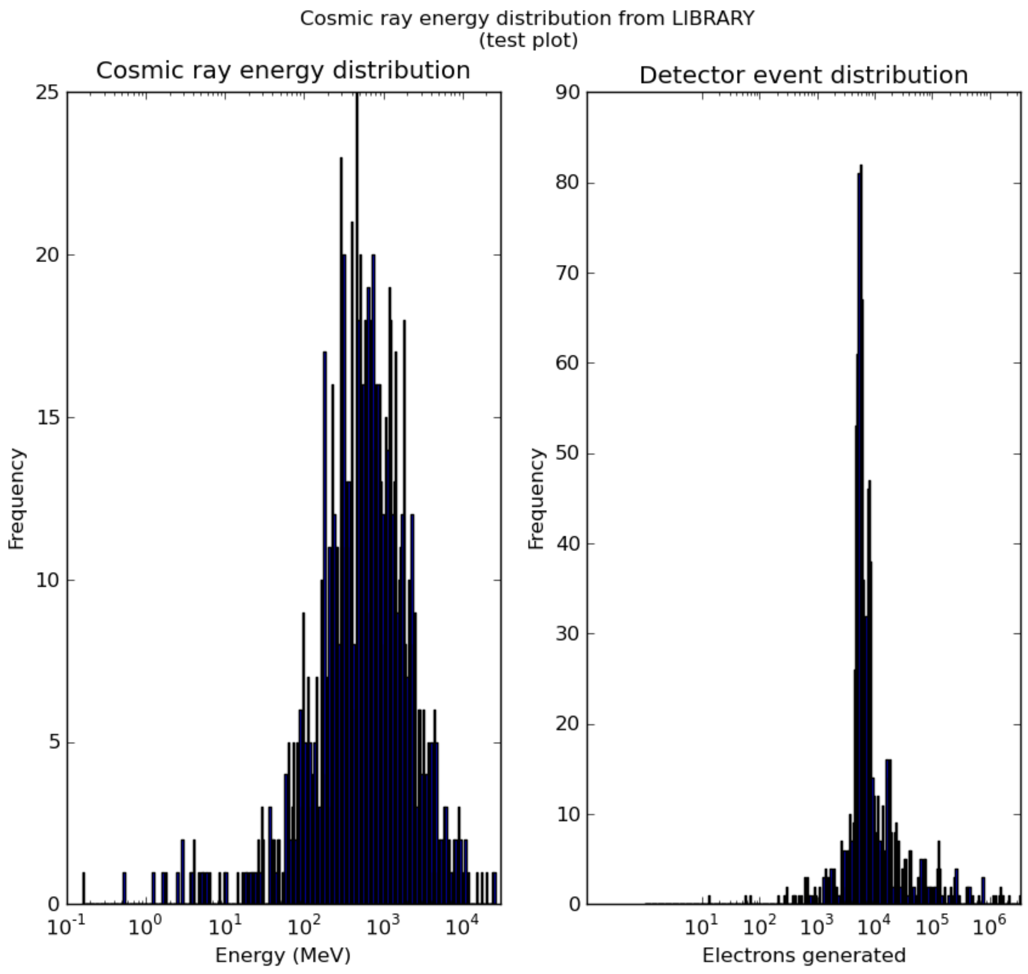

Figure 17: Energy distribution taken from the cosmic ray library

Some typical cosmic ray events are shown in Figure 18. Each image represents one cosmic ray event, and shows the energy dumped into the central pixel and neighbouring pixels. The length and direction of the trail depend on the direction and energy of the incoming particle. The trails tend to be long streaks because the MIRI detectors are thick.

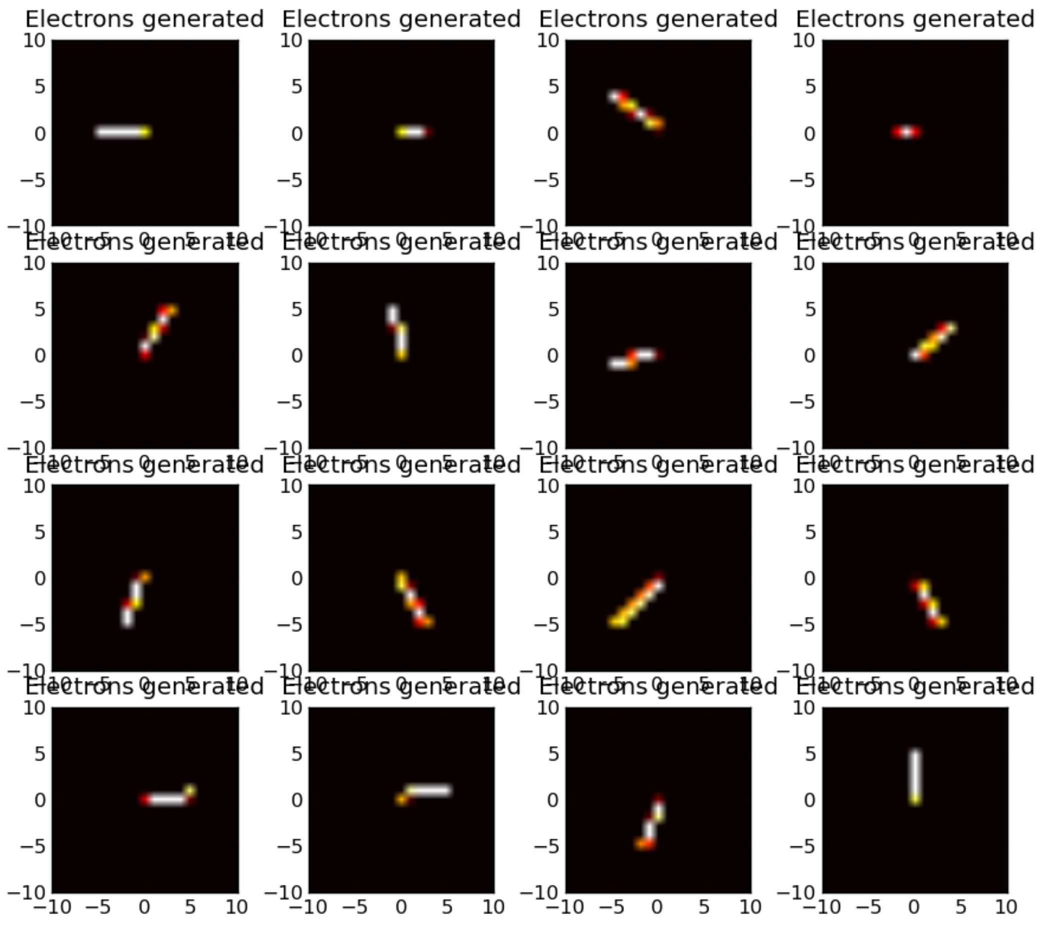

Figure 18: Some typical cosmic ray events

SCASim also attempts to simulate the effect of a cosmic ray striking a readout amplifier. The amplifiers have a much smaller target area than the detectors, so this is a much rarer event. Two effects are assumed:

- If a cosmic ray happens to strike at the moment a pixel is being sampled by the analogue to digital converter (ADC) the resulting value is assumed corrupted. The ADC is then assumed to start a recovery process. If the incoming energy is small the next pixel converted will be ok, but if the incoming energy is large the next pixels may also be affected. This will cause a row of mis-read pixels.
- It is assumed the energy dumped by a cosmic ray will temporarily increase the read noise of an amplifier. All the pixels read by that amplifier will have an enhanced read noise that gradually decays away.

An amplifier's sensitivity to cosmic rays are defined in amplifier_properties.py. See section 8.3.3.

8 DEFINING SCA PROPERTIES

8.1 Cosmic Ray Properties

The properties of the cosmic ray environment are specified in the file scasim/lib/cosmic_ray_properties.py. The properties can be adjusted by copying this file to the current working directory and editing it. The file contains the following variables:

8.1.1 Cosmic ray flux

The following variable declares the cosmic ray flux incident on the MIRI SCA units (in particles per square micron per second).

```
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 variable CR_FLUX_MULTIPLIER is normally set to 1.0, but it can be adjusted to simulate larger or smaller cosmic ray fluxes in all the modes.

8.1.2 Cosmic ray library

Each time a cosmic ray particle hits the SCA its properties are chosen randomly from a pre-defined library of cosmic ray events (see [13]). The library files are specified using this variable:

```
CR_LIBRARY_FILES = {}
CR_LIBRARY_FILES['NONE'] = None
CR_LIBRARY_FILES['SOLAR_MIN']   = find_file_prefix('CRs_SiAs_SUNMIN_')
CR_LIBRARY_FILES['SOLAR_MAX']   = find_file_prefix('CRs_SiAs_SUNMAX_')
CR_LIBRARY_FILES['SOLAR_FLARE'] = find_file_prefix('CRs_SiAs_FLARES_')
CR_LIBRARY_FILES['MIN'] = 0
CR_LIBRARY_FILES['MAX'] = 9
```

Note that the “find_file_prefix” function locates files with the specified prefix. It uses the same search path as for other SCASim configuration files. In fact, there are 10 sets of library files and a set is chosen randomly by choosing a number between 0 and 9 and adding it to the filename as a suffix.

8.1.3 Alternative cosmic ray generation

If the cosmic ray library files are not available, cosmic ray energies are selected randomly from one of the following energy distributions:

```
CR_ELECTRONS['NONE']
CR_ELECTRONS['SOLAR_MIN']
CR_ELECTRONS['SOLAR_MAX']
CR_ELECTRONS['SOLAR_FLARE']
```

each energy distribution contains a list of points giving energy bins (in eV) and the relative probability of a cosmic ray falling in that energy bin.

The variable CR_COUPLING describes the capacitative coupling constant between the detector pixels, which tends to distribute the energy received by a pixel with its immediate neighbours.

8.2 Detector Properties

8.2.1 Detector chip parameters

The properties of the MIRI detectors are specified in the file scasim/lib/detector_properties.py. The properties can be adjusted by by copying this file to the current working directory and editing it. The detectors known to SCASim are defined in this detectors dictionary variable:

```
DETECTORS_DICT =  {'493':_sca493,
                   '494':_sca494,
                   '495':_sca495}
```

This top level dictionary contains a set of lookup keywords (in this case ‘493’, ‘494’ or ‘495’) which can be used to obtain the name of another dictionary describing a particular detector. MIRI has three detectors, as shown above, but this list may be extended to cover more detectors (e.g. for the other JWST instruments). Each detector is defined like the following example:

```
_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
_sca493['CHIP'] = 'SiAs'                   # Type of detector chip
_sca493['COMMENTS'] = "Describes MIRI FPM S/N 106 detector data with ref pixels"
_sca493['ILLUMINATED_ROWS'] = 1024           # Number of illuminated rows
_sca493['ILLUMINATED_COLUMNS'] = 1024    # Number of illuminated columns
_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'] = (1.0e-8, 0.03, 0.0)   # Persistence coefficients
_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)]
_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_current_493')
_sca493['QE_FILE'] find_simulator_file ('qe_measurement_493')
_sca493['BAD_PIXEL_MAP'] = find_simulator_file ('bad_pixels_493')
_sca493['DARK_MAP'] = find_simulator_file ('dark_map_493')
```

Note that the find_simulator_file" function locates files with the specified name. It uses the same search path as for other SCASim configuration files (i.e. current working directory first, then data directories, then PYTHONPATH). The first few parameters define the identify of the detector. The next set of parameters define the size of the detector surface and the number of reference columns. The reference rows are described as they would appear in a level 1 FITS file after all the data manipulation has taken place. The pixel size determines the target area of the detector for cosmic ray hits. The timing parameters determine the exact translation of readout mode into integration time. Finally, the last few parameters give the names of data files describing the dark current, quantum efficiency and bad pixels for this detector. Dark current is described in two different files: one describing how the dark current varies with temperature and the other, the dark map, describing how the dark current varies over the detector surface.

8.2.2 Readout modes

The following dictionary describes the readout modes known to SCAsim:

```
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)
```

Further readout modes may be defined by adding more entries. Each readout mode translates to a list of 8 integer parameters:

- **nsample** – The number of A/D samples making up each readout.

- **discard** – The number of A/D samples discarded before averaging.
- **nframes** – The number of frames averaged per group.
- **groupgap** – The number of frames dropped between groups.
- **ngroups** – The default number of readout groups per integration.
- **nints** – The default number of integrations per exposure.
- **avggrps** – The number of groups averaged to reduce data rate.
- **avgints** – The number of integrations averaged to reduce data rate.

8.2.3 Subarray modes

The following dictionary describes the subarray modes known the SCAsim:

```
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)
```

Further subarray modes may be defined by adding more entries. Each subarray mode translates to a list of 4 integer parameters:

- **firstrow** – The starting row for the subarray on the detector surface.
- **firstcol** – starting column for the subarray on the detector surface.
- **subrows** – The height of the subarray in rows.
- **subcolumns** – The width of the subarray in columns.

8.2.4 Cosmic ray sensitivity

Finally, the variable COSMIC_RAY_LEAKAGE_FRACTION determines what fraction of cosmic ray hits result in a downwards jump, as described in [12].

~~8.3 Amplifier Properties~~

~~8.3.1 Amplifier parameters~~

~~The properties of the MIRI detector readout amplifiers are specified in the file scasim/lib/amplifier_properties.py. The properties can be adjusted by by copying this file to the current working directory and editing it. The amplifiers known to SCASim are defined in this amplifiers dictionary variable:~~

~~AMPLIFIERS_DICT = {'494':_amplist494,~~
~~'495':_amplist495,~~
~~'493':_amplist493}~~

~~which itself translates into the following lists of amplifiers:~~

~~_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)~~

~~Each amplifer list defines the amplifiers associated with a particular detector. In the case of MIRI, each detector has five amplifiers (four of which read the detector pixels and a fifth used as a reference output). Just as with the detectors, each amplifier is described by a dictionary looking like this:~~

~~_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['MAX_DN'] = 65535.0            # Maximum DN output by this amplifier~~
~~_amp493_1['READ_NOISE_FILE'] = find_file('read_noise', '493', '1')~~
~~_amp493_1['READ_NOISE_COL'] = 1    # Column number where to find read noise~~

~~_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['MAX_DN'] = 65535.0            # Maximum DN output by this amplifier~~
~~_amp493_5['READ_NOISE_FILE'] = find_file('read_noise_493')~~
~~_amp493_5['READ_NOISE_COL'] = 5    # Column number where to find read noise~~
```

~~The first few parameters define the identify of the amplifier (note that the name is made up of the detector name plus a sequence number). The TYPE parameter gives the amplifier type, which may be “illuminated” if it is a normal amplifier reading the detector pixels or “reference” if it is a reference output amplifier. The REGION parameter gives the starting column for a normal amplifier or a choice of top/bottom region for a reference output amplifier. (NOTE: The reference pixels for MIRI are always as the top. The “bottom” option exists for future flexibility.) The amplifier area is used to define its probability of being hit by a cosmic ray (set ot to zero if you want to exclude cosmic ray effects altogether). The BIAS, GAINTYPE and GAIN describe the amplifier’s readout properties. The bias level is added to all readings and the gain function used to convert electrons into DN. The gain function may be a polynomial or a linear scale factor. Finally, the READ_NOISE parameters give the name of file which describes how the amplifier read noise varies with temperature. Several amplifiers may be described within one file, so the last parameter gives the column number relevant to this particular amplifier.~~

~~8.3.2 Amplifier simulation control~~

~~The following two parameters control how short term drifts in the amplifier~~
~~# output are simulated. They give the initial reference level in electrons and the maximum random drift in the reference level expected after each readout. Set them to zero to turn off this simulated effect.~~

```
~~INITIAL_REF_LEVEL = 5~~
~~MAX_REF_DRIFT = 3~~
```

~~8.3.3 Cosmic ray sensitivity~~

~~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 generates 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.~~

~~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.~~

~~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.~~

~~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.~~