The use of Python to simulate the behaviour of the Mid Infrared Instrument on JWST

J. Morin[1] and the MIRI European Consortium Software Team

R. Azzollini[2], S. Beard[3], J. Blommaert[4], P. Bouchet[5], J. Bouwman[6], B. Brandl[7], C. Cavarroc[8], S. Chaintreuil[9], R. Gastaud[5], A. Glasse[3], A. Glauser[3], R. Huygen[4], F. Lahuis[10], O. Littlejohns[11], C. Nehme[5], J. Pye[11], T. Ray[1], A. Scaife[1], J. Schreiber[6], B. Vandenbussche[4]

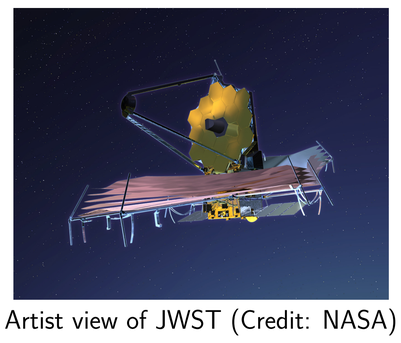
[1]DIAS (Ireland), [2]CSIC (Spain), [3]UKATC (UK), [4]KUL (Belgium), [5]CEA (France), [6]MPIA (Germany), [7]Univ. Leiden (Netherlands), [8]IAS (France), [9]LESIA (France), [10]SRON (Netherlands), [11]Univ. Leicester (UK)

Contacts: jmorin@cp.dias.ie, F.Lahuis@sron.nl, steven.beard@stfc.ac.uk

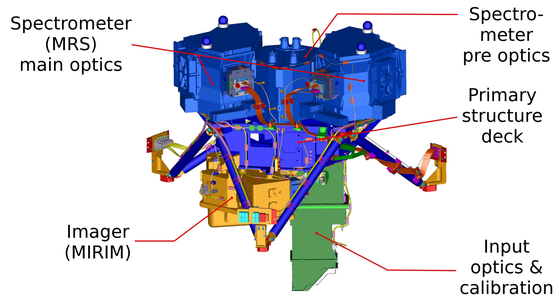
Abstract

Python has been elected as the main language for software aimed at reduction and analysis of data generated by the James Webb Space Telescope. We present the development environment set up for the Mid-Infrared Instrument and its use to implement data simulators.

JWST/MIRI instrument concept



Artist view of JWST (Credit: NASA)



MIRI OBA and key subsystems (Copyright: University of Leicester, UK)

The James Webb Space Telescope (JWST) is 6.5 meter space telescope optimized for infrared wavelengths due to be launched at the end of the decade. The Mid-Infrared Instrument (MIRI) is one of the 4 instruments designed for JWST [1]. **MIRI is a very versatile instrument** operating in the 5-28 μm wavelength range and divided in two optical channels. The first channel provides direct imaging, coronagraphic imaging at selected wavelengths and long-slit low-resolution spectroscopy (LRS, R= $\frac{\lambda}{\delta\lambda}$ ~100). The second one can record a data cube of medium resolution spectra (MRS, R~3,000) through an integral field unit.

JWST/MIRI Software development environment

Development of data reduction and analysis software as well as simulators for JWST/MIRI is a joint effort between STScI and MIRI European Consortium, now based on Python. The availability of a number of high-quality libraries and tools and of solutions to interface with compiled code make **Python an excellent alternative to data-oriented proprietary scripting languages** such as IDL.

Development environment

Libraries

- Numpy
- SciPy
- matplotlib
- stsci_python
- astrolib

Development tools

- sphinx
- unittest
- setuptools
- pylint

Scripts and templates

- Install w/ package selection and dependency checks
- Documentation build
- Templates for code, documentation and tests

Code repository architecture

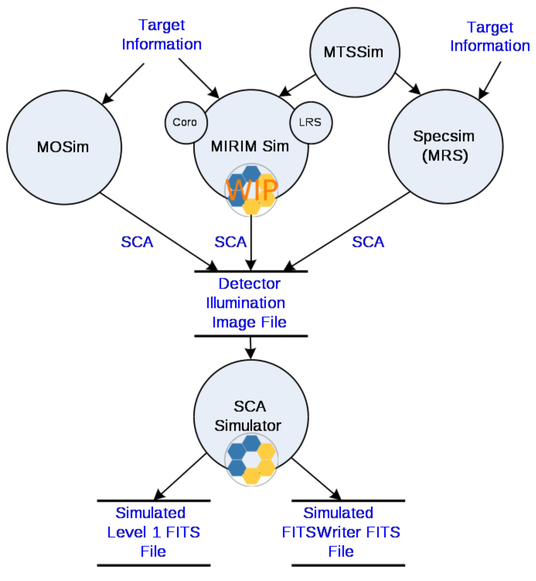
- Identical for all packages
- Separates code from docs, tests, scripts, data
- Provides namespaces
- and a common tools package

Due to the distributed nature of our code development, the will to explore alternative processing methods, and the foreseen evolution of instrument knowledge, a **flexible development strategy** is required. Our development environment [2] – schematically described above – largely relying on standard Python tools reveals helpful to achieve such flexibility.

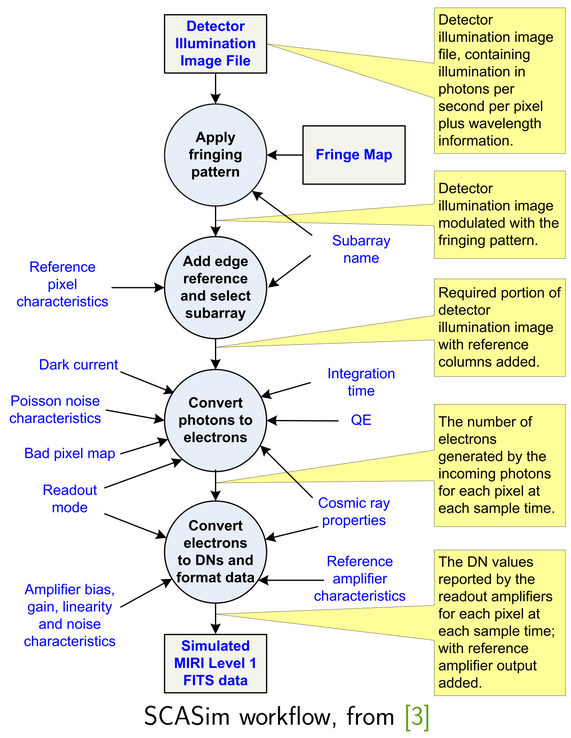
Simulators workflow

Data simulators are useful throughout the instrument development and exploitation: from pre-launch test support and reduction pipeline tests, to observation planning. These simulators need to be **adaptable** so that information learned about the instrument during the pre-launch testing and in-orbit operation can be fed back into the simulation. MIRI simulators are divided in three parts which operate sequentially:

1. **MTSSim** calculates the **irradiance** of the **MIRI Telescope Simulator** which is the input source for the instrument test and ground calibration campaigns.
2. **MOSim**, **MIRIMSim** or **Specsim** simulate an **astronomical target** (spatial and spectral characteristics) and the **optical train** to produce an illumination map of the detector.
3. **SCASim** simulates the **behaviour of the detector** (or SCA – Sensor Chip Assembly) including generic features and some more specific to MIRI (e.g., non-destructive readouts, Poisson statistics).



MIRI simulators workflow, from [3]

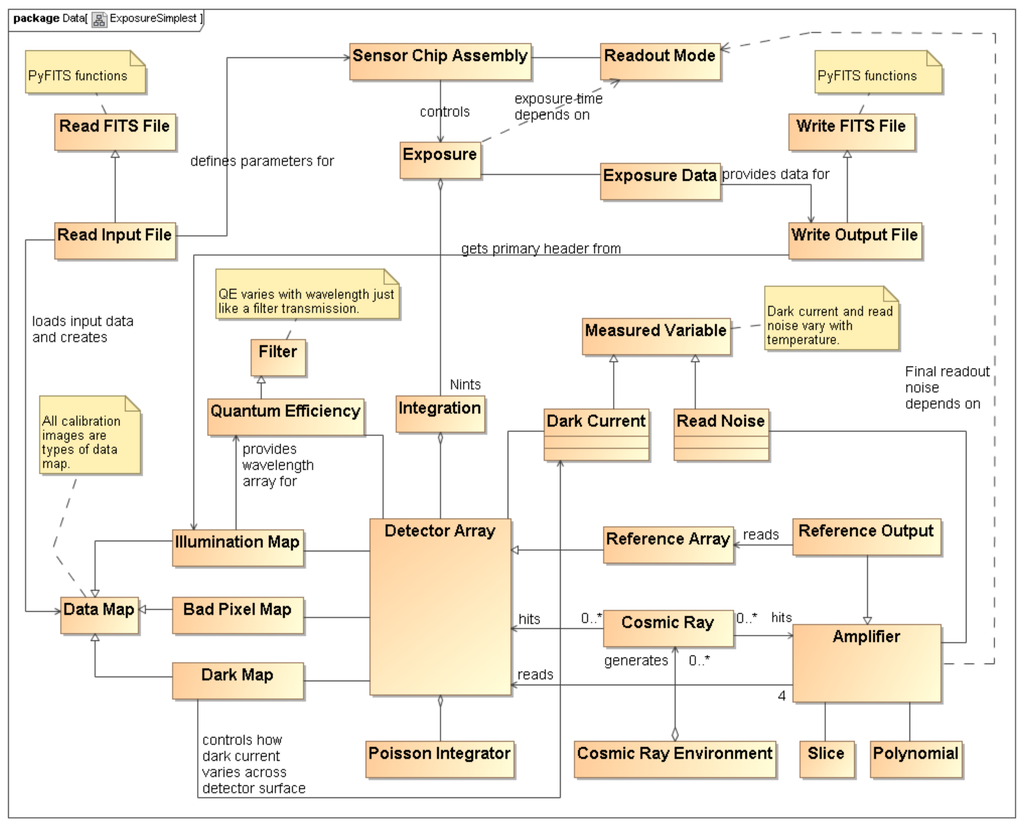


SCASim workflow, from [3]

Presently only SCASim is based on our Python development environment, MIRIMSim port is ongoing work.

SCASim design and implementation

Object-oriented (OO) design is efficient for simulators, the knowledge of an instrument component or property is encapsulated in a class from which a number of instances or subclass instances can be derived. **Separation of the actual processing code from parameters** is efficiently achieved in this design. The core of SCASim is the Detector Array (see diagram below and [3]), which represents a collection of detector pixels, and aggregates a number of objects that manage its main properties (either static or dynamically evolving during a simulation) such as health, quantum efficiency or dark current. The OO design and Numpy array features are also well-suited to deal with the variety of possible I/O formats.



SCASim class diagram, adapted from [3]

Assessment of our Python experience

- Switching to Python/Numpy/SciPy is rather easy thanks to "natural" syntax.
- Python and Numpy encourage writing readable and concise code.
- Array processing performance is good, provided appropriate syntax is used.
- Dictionaries are a powerful feature for storing configuration data directly in Python files.
- SCASim works on various GNU/Linux flavours, Mac OS X, Win XP/7 and Python 2.5–2.7 with minimal adjustments.
- The weak typing of Python is rather unusual for scientists but provides the required flexibility, in particular when combined to other language features such as optional arguments/default values.

References

[1] "The JWST MIRI instrument concept", G.S. Wright et al. 2004, Proc. SPIE, 5487, 653
[2] "MIRI Software Development Plan", F. Lahuis et al., SRON, 24th September 2010
[3] "SCASim Software Design Document" S. Beard, UKATC, 12th January 2011

Acknowledgements

J. Morin gratefully acknowledges the funding support of the ESA/PRODEX program. We acknowledge fruitful collaboration with our colleagues of the STScI SSB for JWST/MIRI software development, and are thankful to all contributors of the FOSS Python ecosystem.

MIRI Software resources

- SVN repository: `https://svn6.assembla.com/svn/jwst`
- SCASim installation instructions: `www.dias.ie/~jmorin/miri/install.html`