

Using Perl in Basic Science and Calibration Pipelines for Spitzer Infrared Array Camera Data

H. Brandenburg, P. Lowrance, R. Laher, J. Surace, and M. Moshir

*Infrared Processing and Analysis Center, Caltech, Pasadena, CA 91125,
Email: heidi@ipac.caltech.edu*

Abstract. Object oriented Perl language pipelines generate calibration products and basic calibrated data from raw images taken by the Infrared Array Camera (IRAC) onboard NASA's Spitzer Space Telescope. The pipelines gather input data and control files, initiate database interactions, and manage data flow through C, C++, and Fortran component programs. The compiled component programs perform instrumental signature correction, calibration, and data characterization.

Core pipeline functionality is provided by two compact Perl object hierarchies - one for pipelines and another for images. The objects allowed flexible and agile response to change during Spitzer's first year of operations, which offset the cost-impact of utilizing interpreted Perl on production data processing.

1. Execution Environment

Data processing takes place on a cluster of low-end Solaris servers ("pipeline drones"); the drones may run multiple, possibly unrelated, jobs concurrently. The Pipelines write products to a disk on their host, then copy the final products to a network drive at the end of processing.

Job preparation and control software (written in C), under the control of an executive, prepares directories, seeds a FITS file, and then initiates and waits for the pipeline.

Most intraprogram communication occurs via environment variables. Pipelines communicate with the science operations database primarily through a Perl API. Calibration files and control data files (CDFs, or resource files) are written directly to a pipeline's working directory.

2. Classes & Methods

Two class hierarchies provide most of the required pipeline functionality (Figure 1). Each hierarchy is rooted in an `_Initializable` container class which constructs a new hash-based object and blesses it into its derived class, and then calls an `_init()` method on that class to force object initialization to begin in the derived class and bubble up through parent classes.

2.1. Image

By SSC convention all images are assumed FITS formatted. The Image parent class provides read-only header keyword access.

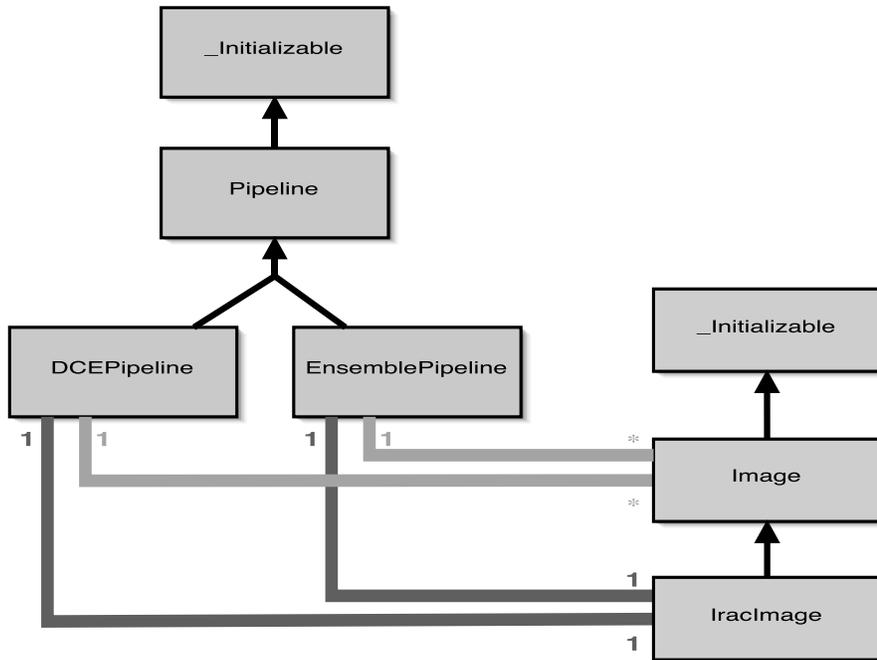


Figure 1. Class Relationship and Inheritance

2.2. Image::IracImage

Image::IracImage contains data and methods specific to IRAC observing. Its private methods include image integrity and telemetry checks. Public methods provide access to observing parameters (frametime, fowler number, wait periods, etc.), assembly of calibration transfer parameters, and pack/unpack functionality for sub-array data.

2.3. Pipeline

The Pipeline class contains a hash holding the current pipeline state, including the current input file name, the next output file name, and where to find calibration data and other control file resources. The class contains private methods for environment checks, parsing control files, and database interactions. It also contains public methods to allow the pipeline scripts to access and mutate the state hash, load control files, execute programs, and end the pipeline.

2.4. Pipeline::DCEPipeline

The DCEPipeline class contains functionality specific to single image pipelines. It also implements public methods for single DCE related database interactions.

2.5. Pipeline::EnsemblePipeline

The EnsemblePipeline class contains functionality specific to processing sets of images together. It extends the basic pipeline state hash with ensemble-related

items and provides accessor and mutator methods for them. The class also provides additional public methods for ensemble-related database interactions.

Pipeline objects require an image object for instantiation. During pipeline processing, messages may be passed from Pipeline objects to Image objects.

3. Pipeline Scripts

The driving design aesthetic in the launch era IRAC pipeline was to create a simple code base with no redundancy in order to minimize labor and risk during change. Extant pipelines for calibration product processing and basic science reduction total around 2000 source lines. Each of eight pipeline scripts contain between 100 and 550 lines of executable perl source in a single file.

All pipelines begin by instantiating an `IracImage` object with the FITS file seeded by the job preparation and control program. The image object is then used to instantiate a pipeline object (the image becomes part of a pipeline's internal state). The pipeline script calls a public method on the pipeline object to load control data (resource) files.

Pipeline scripts then execute compiled component programs for instrumental signature correction, data characterization, and data calibration (Figure 2). When numerical processing is finished, the pipeline registers products with the science operations database and moves them into a network file system.

Detailed pipeline and module behavior is changed by configuring a control data file. For example, all calibration observations are preprocessed by the same pipeline script, but darks and flats are processed with parameter sets appropriate to each kind of observation (e.g., Figure 3).

4. In Operations

There is an intrinsic cost to utilizing an interpreted language like Perl for pipeline processing. Although we have not yet quantified the percent of pipeline time spent by the interpreter or in process bookkeeping for the modular infrastructure (anecdotal evidence from very early phases of development indicated that the overhead was in the few percent range), we have met the project IRAC processing throughput requirements -using nearly outdated Netra-class hardware in operations. In particular all data are processed at least 5 times faster than the rate of data acquisition.

Nearly as important, the Perl code allows the easiest response to change. To build and deploy a compiled patch, change management procedures require a new release, increasing workload and turnaround time. Since SSC Perl scripts and modules do not require a build step for configuration and install on SSC systems, Spitzer staff were able to support a larger number of changes during in-orbit checkout than would have been possible with compiled code.

Acknowledgments. This work was carried out at the Spitzer Science Center, with funding from NASA under contract to the California Institute of Technology and the Jet Propulsion Laboratory.

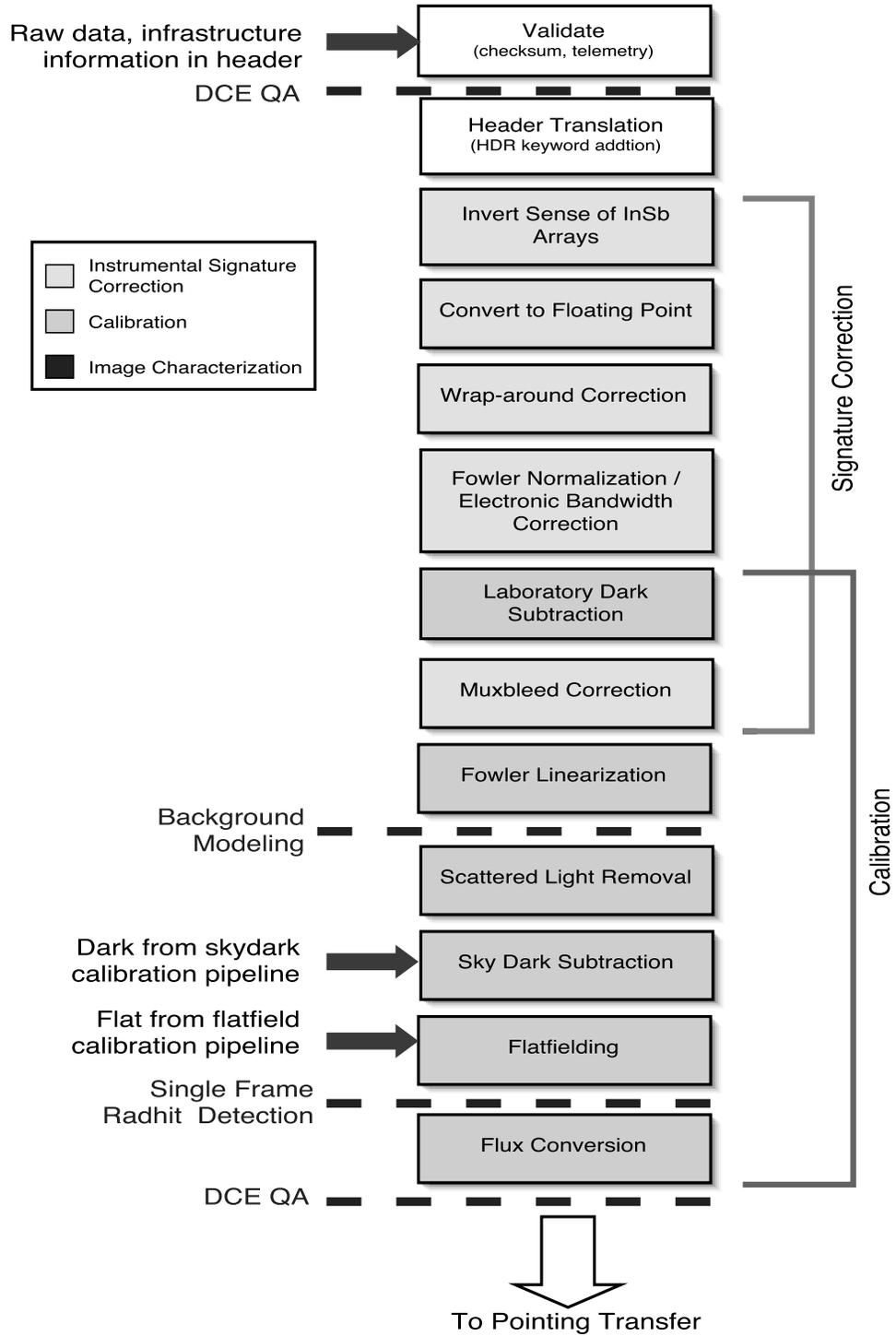


Figure 2. Science (Basic Calibrated Data) Pipeline

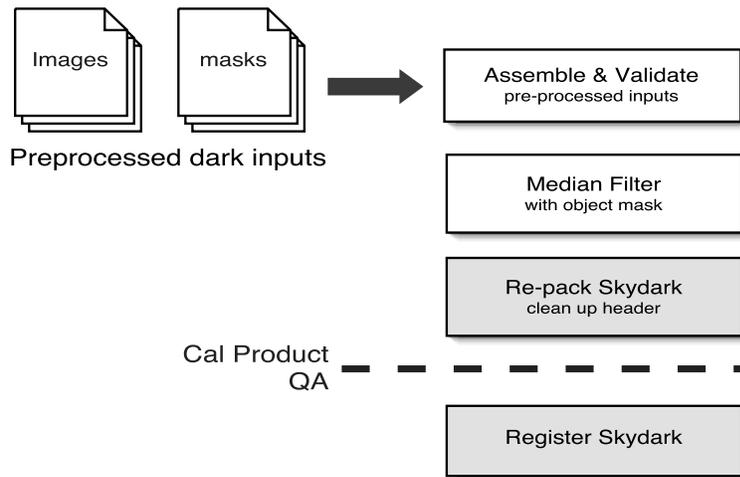


Figure 3. Skydark Creation Ensemble Pipeline