Large Synoptic Survey Telescope

Abstracting the storage and retrieval of image data at the LSST

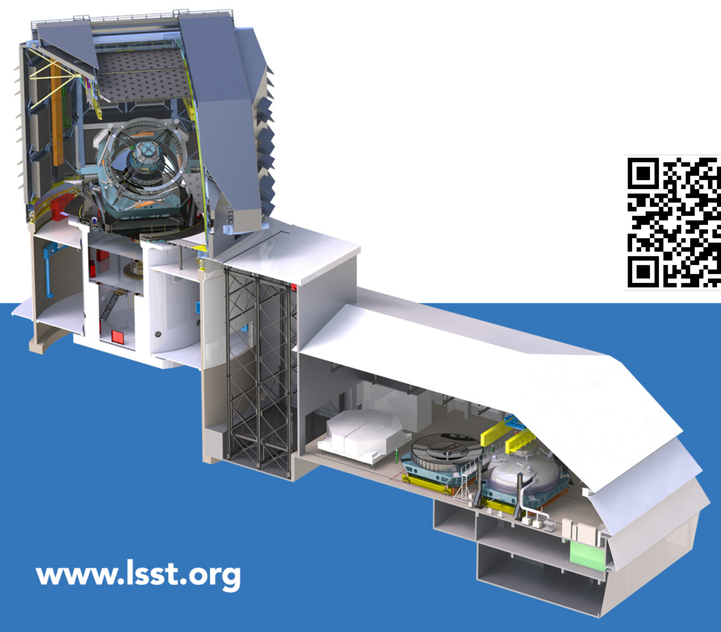
Tim Jenness (AURA/LSST), James Bosch (Princeton), Pim Schellart (Princeton), K-T Lim (SLAC), Andy Salnikov (SLAC), Michelle Gower (NCSA)

INTRODUCTION

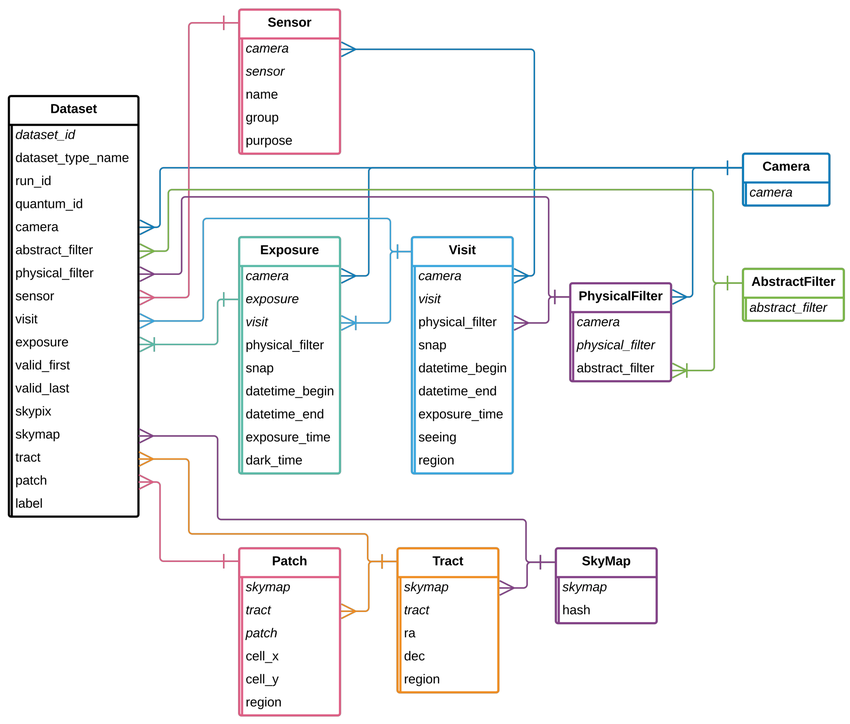
Writing generic data processing pipelines requires that the algorithmic code does not ever have to know about data formats of files, or the locations of those files. At LSST we have a software system known as "the Data Butler", that abstracts these details from the software developer. Scientists can specify the dataset they want in terms they understand, such as filter, observation id, date of observation, and instrument name, and the Butler translates that to one or more files which are read and returned to them as a single Python object. Conversely, once they have created a new dataset they can give it back to the butler, with a label describing its new status, and the butler can write it in whatever format it has been configured to use. All configuration is in YAML and supports standard defaults whilst allowing overrides.

BUTLER COMPONENTS

The Butler consists of three core components: Schema, Registry access, and Datastore. The Schema defines the data model for relating datasets to each other, and is defined consistently for all datasets. This allows you to ask which datasets do I need to calibrate this one, or which datasets were taken with this filter between these dates. The Registry classes allow the data model to be queried and are configurable via plugins to allow different backend representations of the data. Finally the Datastore deals with the reading and writing of datasets themselves. There are datastores for a POSIX file system, an in-memory cache, and chained datastores (where writes go to all datastores and reads pull from the first datastore to return it). We intend to soon add support for object stores. To support the Datastores "Formatters" have to be written to serialize and deserialize Python objects.

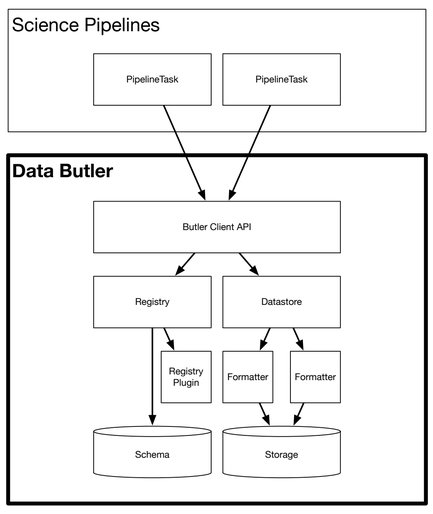


BUTLER DATA MODEL



The Butler data model is designed to understand the relationships between observations and calibrations, but also the how the sky can be segmented into different regions. Each observation is linked to corresponding patches on the sky map. These are used to easily get an answer to the question of which datasets should be included for a coadd of a particular sky region.

BUTLER IN USE



Individual pipeline tasks work with Python objects. They put datasets and retrieve datasets from the Datastores. The Butler maps a Python object to a serialization format through a "StorageClass" defined in the YAML configuration files. Changing the serialization format from FITS to HDF5 does not require any code changes for the user. Pre-defined components of a dataset, such as the WCS solution, can be retrieved without reading the full dataset if supported by the formatter.

Below is some user code for retrieving a calibrated exposure of an HSC observation, transforming it, and then storing a new version with a different dataset type name.

```
from lsst.daf.butler import Butler

butler = Butler("config.yaml")

dataId = {"instrument": "HSC", "obsid": "HSCA04090000"}
calibrated = butler.get("calexp", dataId)
warped = doWarp(calibrated)
butler.put(warped, "warpexp", dataId)

wcs = butler.get("warpexp.wcs", dataId)
```

HEADER TRANSLATION

In order to be able to ingest instrument data into a Butler repository the Butler has to understand some properties of the instrument including filters, detector information, and how to extract metadata from data headers. We have written a separate Python package, astro_metadata_translator, to support header translation and metadata extraction for astronomical instrument headers. New translators must be written to allow the Butler to understand data during ingest. Currently translators exist for DECam, CFHT MegaPrime, and SuprimeCam and Hyper-SuprimeCam from Subaru. This package solely depends on Astropy and does not need any LSST infrastructure.

```
from astropy.io import fits
from astro_metadata_translator import ObservationInfo

hdul = fits.open("hsc.fits")
obsInfo = ObservationInfo(hdul[0].header)
print(f"instrument={obsInfo.instrument}, "
      f"date-obs={obsInfo.datetime_begin}")
```

SUMMARY

The Butler frees you from the worry of file formats and file systems when your main concern is processing and characterizing datasets. The Butler system is not LSST-specific and is entirely driven by external configuration to suit a specific use case. The Butler will be released at the end of the year alongside v17.0 of the LSST Science Pipelines.

More Information

Butler documentation: https://pipelines.lsst.io/v/daily/packages/daf_butler/
Schema overview: https://dmtn-073.lsst.io
LSST Data Management Overview: arXiv:1512.07914
LSST Design Overview: arXiv:0805.2366

www.lsst.org

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreement 1258333 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory. Additional LSST funding comes from private donations, grants to universities, and in-kind support from LSSTC Institutional Members.