Large Synoptic Survey Telescope

www.lsst.org

Porting the LSST Data Management Pipeline Software to Python 3

Tim Jenness *(LSST Tucson)* for the LSST Data Management Team

Abstract The LSST data management science pipelines software consists of more than 100,000 lines of Python 2 code. LSST operations will begin after support for Python 2 has been dropped by the Python community in 2020, and we must therefore plan to migrate the codebase to Python 3. During the transition period we must also support our community of active Python 2 users and this complicates the porting significantly. We have decided to use the Python `future` package as the basis for our port to enable support for Python 2 and Python 3 simultaneously, whilst developing with a mindset more suited to Python 3. In this poster we report on the current status of the port and the difficulties that have been encountered.

The LSST Software Stack

The Large Synoptic Survey Telescope will take about 15 TB of image data per night, amounting to 0.5 exabytes of image data and 15 petabytes of catalog data after ten years of operation. We are writing a suite of software packages to enable these data products to be created with sufficient quality and performance to meet the established science goals. The science pipeline software enables two key components of the data management system. The Alert Production pipelines (also known as *Level 1*) process the data from the telescope and publish alerts to the community within 60 seconds of data acquisition. The Data Release Production (*Level 2*) pipelines are responsible for the annual data releases which reprocess all the data each year to generate the best possible catalogs. Both these systems are integrated with the Calibration Products pipeline that continuously calculates the best calibrations. The software also provides a toolkit for user-supplied code that can be used to efficiently and effectively analyze LSST data as part of *Level 3* processing or their own Pipelines. LSST software development began in 2004 (Python 2.3) and still has a lot of code that was written in the 2.4 to 2.6 era.

Why support Python 3?

1. The official statement from the Python developers is that Python 2.7 support will be dropped in 2020. This is before the official start of the LSST survey and we do not want to commission software where the key executable under-pinning the entire system will soon lose support.
2. LSST has external users of our software that provide early beta testing services and we do not want to actively impede those users from migrating to a more modern Python.
3. Astropy recently declared that it would stop adding new features to Python 2-compatible releases in 2017 (APE10), joining Ipython, Sunpy, and matplotlib. This declaration will motivate the community, and furthermore, LSST DM recently decided to integrate Astropy into our pipelines code.

Stack Components

The full list of stack packages are listed to the right, along with the current porting status.

daf The Data Access Framework is responsible for mediating between the archive resources and the application writer. The pipeline code has a completely abstract view of file I/O and only has to know how to deal with data objects representing fundamental types such as exposures and tables. Currently FITS is the internal format but the system is designed such that the internal format could be changed to HDF5, for example, and no changes would have to be made to the science pipeline code. This abstraction of the files from the code protects us against shifts in format preferences.
dax Data access libraries.
afw The Astronomy FrameWork provides the core classes for manipulating images and catalogs, including detecting sources and world coordinate handling.
ip These are the image processing classes, including packages for instrument signature removal and image differencing.
meas The measurement packages include code for determining source properties and correcting astrometry and photometry.
obs These classes provide instrument-specific knowledge to the software system, providing information to the data access framework to teach it how to interpret data from a range of optical cameras. The *obs* packages currently support data from DECam and a selection of instruments on Subaru and CFHT.
pex Pipeline execution support.
pipe Pipeline infrastructure and tasks. A task is the name for a core processing component that can be chained with other tasks to build a pipeline.

- ✅ afw
- ✅ base
- ✅ cat
- ❌ ci_hsc
- ✅ coadd_chisquared
- ✅ coadd_utils
- ✅ ctrl_events
- ✅ ctrl_execute
- ✅ ctrl_orca
- ✅ ctrl_pool
- ✅ ctrl_provenance
- ✅ ctrl_stats
- ✅ daf_base
- ✅ daf_butlerUtils
- ✅ daf_persistence
- ❌ datarel
- ❌ dax_dbserv
- ❌ dax_imgserv
- ❌ dax_metaserv
- ❌ dax_webserv
- ❌ dax_webservcommon
- ✅ db
- ✅ display_ds9
- ✅ geom
- ✅ Ip_diffim
- ✅ Ip_isr
- ✅ log
- ✅ meas_algorithms
- ✅ meas_astrom
- ✅ meas_base
- ✅ meas_deblender
- ✅ meas_extensions_ngmix
- ✅ meas_extensions_photometryKron
- ✅ meas_extensions_psfex
- ✅ meas_extensions_shapeHSM
- ✅ meas_extensions_simpleShape
- ❌ meas_modelfit
- ✅ ndarray
- ❌ obs_cfht
- ❌ obs_decam
- ❌ obs_lsstSim
- ✅ obs_monocam
- ❌ obs_sdss
- ❌ obs_subaru
- ✅ obs_test
- ✅ pex_config
- ✅ pex_exceptions
- ✅ pex_logging
- ✅ pex_policy
- ✅ pipe_base
- ✅ pipe_drivers
- ✅ pipe_supertask
- ✅ pipe_tasks
- ❌ Qserv
- ❌ scisql
- ✅ shapelet
- ✅ skymap
- ✅ skypix
- ✅ sphgeom
- ✅ utils

Supporting Python 3

A key requirement for this initial port was that our pipeline code that is used by external users must support both Python 2.7 and Python 3. The Python community has developed a number of schemes for handling this and we looked at both `six` (used by Astropy) and `future`. We decided on future because the resulting code looks almost exactly like Python 3 code, in many cases the code can run on Python 3 without future being installed.

- Clean up code with `autopep8`. Important to remove tabs from code and simplifies lint checking later on.
- Run stage 1 `futurize` to modernize the code to Python 2.7 standard (print function, absolute imports, modern exception catching, update `has_key()` usage).
- Run stage 2 `futurize`. This does a normal `2to3` modernization and adds compatibility imports for Python 2. Fixups include conversion of map(filter(lambda…) constructs to list comprehensions.
- Handle bytes/str interchange by adding decode/encode calls where appropriate. Python 3 is very good at finding inconsistencies.
- Once everything works on Python 3, run it on Python 2. It will probably have issues. Iterate.
- Run full stack integration tests with Python 2 as the lower level libraries are ported.

Porting Issues

The porting process was relatively straightforward, but there were some issues that required thought.

Lists: Some functions used to return lists and now return iterators or views. Add explicit `list()` constructors when needed. Also, use: `for i in mydict` when looping over dictionary keys rather than `for i in list(mydict.keys())`.

Bytes vs Strings: Python 2 is very relaxed about bytes and strings. Python 3 insists on them being distinct. Remember to decode the bytes from external commands and to use binary in pickle files. Enable Unicode support in SWIG interfaces.

```
results = subprocess.check_output(["ls"]).decode()
```

Future str: The future package brings in a `str` object that emulates a Python 3 str. This can cause unexpected trouble if code ever checks if the supplied string is an instance of str. We have imported `basestring` to handle this (an alias for `str` in Python 3) but if your code is not using any Unicode features it is probably better not to use the future implementation.

Long integers: Python 3 always uses long integers and does not support `L` syntax for literals. Our usage was to distinguish 32-bit from 64-bit integers but on a 64-bit Python this is no longer necessary. Remove `long()` and no longer use Python integer type to select C++ integer type.

Metaclasses: These are handled fine by future until multiple inheritance is employed with each parent class using distinct metaclasses.

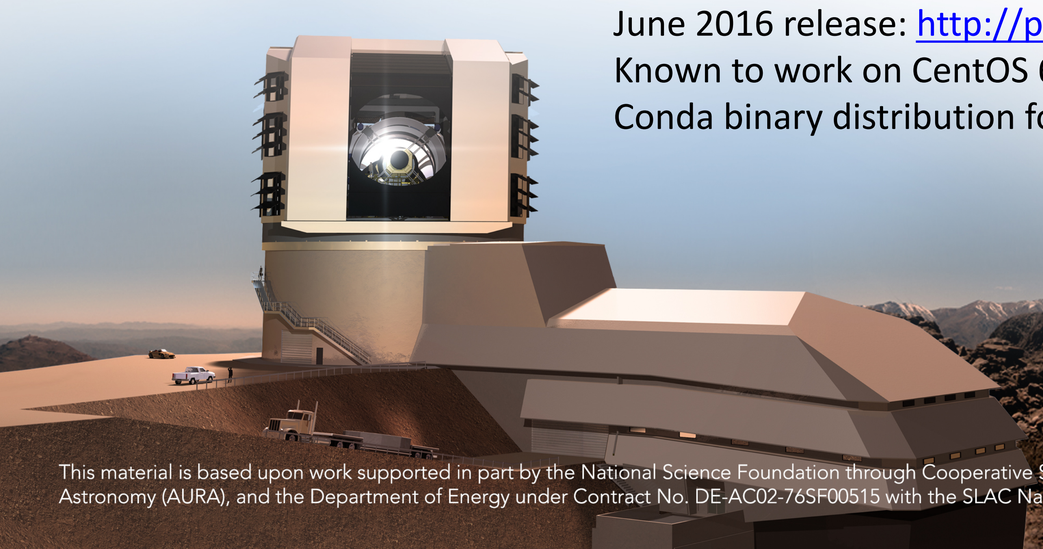
```
if sys.version_info[0] >= 3:
    class _PolicyMeta(type(collections.UserDict), type(yaml.YAMLObject)):
        pass

    class _PolicyBase(with_metaclass(_PolicyMeta, collections.UserDict,
                                     yaml.YAMLObject)):
        pass
else:
    class _PolicyBase(collections.UserDict, yaml.YAMLObject):
        pass
```

Explicit version checking: Reading Python 2 pickle files in Python 3 may need the encoding argument (not supported on Python 2). Any test that is disambiguating bytes and strings returned from a C++ interface should be skipped on Python 2.

Current Status

As of October 2016 approximately 45 packages (see table) have been ported to Python 3 (not including third-party packages or packages without any Python code). Eight science pipeline packages remain to be ported. The aim is for the port to be completed by the end of the year and for the data access libraries and Qserv to be done shortly thereafter.



Obtaining the Software

Source code: https://github.com/LSST
June 2016 release: http://pipelines.lsst.io/releases/notes.html
Known to work on CentOS 6 and 7 & OS X Yosemite & above.
Conda binary distribution for Linux and OS X.

More Information

LSST DM Python 3 Porting Guide: https://sqr-014.lsst.io
LSST Data Management Overview: arXiv:1512.07914
LSST Design Overview: arXiv:0805.2366
LSST Science Book: arXiv:0912.0201
LSST Data Products: http://ls.st/LSE-163
DM Applications Design: http://ls.st/LDM-151
Key Numbers: http://lsst.org/scientists/keynumbers

This material is based upon work supported in part by the National Science Foundation through Cooperative Support Agreement (CSA) Award No. AST-1227061 under Governing 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.