Draft version October 1, 2019
Typeset using LaTeX **twocolumn** style in AASTeX62

# The LSST Pipelines Software

Tim Jenness[1]

[1]*LSST Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA*

## ABSTRACT

The Large Synoptic Survey Telescope Data Management System

*Keywords:* Astrophysics - Instrumentation and Methods for Astrophysics — methods: data analysis — methods: miscellaneous

## 1. INTRODUCTION

The Large Synoptic Survey Telescope (LSST; **?**) is an 8.4 m telescope being built on Cerro Pachon in Chile. The Data Management System (DMS; **?**) is designed to handle the flow of data from the telescope, approaching 20 TB per night, in order to issue alerts and to prepare annual data releases. A central component of the DMS is the LSST Science Pipelines software that provides the algorithms and frameworks required to process the data and generate the coadds, difference images, and catalogs to the user community for scientific analysis.

The LSST Science Pipelines software consists of the building blocks required to construct high performance pipelines to process the data from LSST. It has been under development since at least 2004 (**?**) and has evolved significantly over the years as the project transitioned from prototyping (**?**) and entered into formal construction (**?**). The software is designed to be usable by other optical telescopes and this has been demonstrated with Hyper Suprime Cam (**?**).

In this paper we provide an overview of the components of the software system. This includes a description of the support libraries and data access abstraction, along with the algorithmic components and the pipeline task system. We do not include validation of the individual algorithms. The other components of the LSST DMS, such as the workflow system, the Qserv database (**?**) and the LSST Science Platform (**?**), are described elsewhere.

## 2. FUNDAMENTALS

The LSST Science Pipelines software is written in Python with C++ used for high performance algorithms and for core classes that are usable in both languages. We recently dropped Python 2 and adopted Python 3 (**?**), requiring a minimum version of Python 3.6. The C++ layer can use C++14 features and we use pybind11 to provide the interface from Python to C++. Additionally, the C++ layer uses `ndarray` to allow seamless passing of C++ arrays to and from Python `numpy` arrays. This compatibility with `numpy` is important in that it makes LSST data structures available to standard Python libraries such as Scipy and Astropy (**??**).

Although all the software uses the `lsst` namespace, the code base is split into individual Python products in the LSST GitHub organization[1] that can be installed independently and which declare their own dependencies. These dependencies are managed using the EUPS system (**??**). Each product is built using the SCons system (**?**) with LSST-specific extensions provided in the `sconsUtils` package enforcing standard build rules.

For logging we use Log4CXX wrapped in the `lsst.log` package to make it look more like standard Python logging whilst also supporting deferred string formatting such that log messages are only formed if the log message level is sufficient for the message to be logged. Finally, we also provide some LSST-specific exceptions that can be thrown from C++ code and caught in Python.

As of April 2018, the Science Pipelines software is approximately 290,000 lines of Python and 225,000 lines of C++.[2]

### 2.1. *Unit Testing and Code Coverage*

Unit testing and code coverage are critical components of code quality (**?**). Every package comes with unit tests written using the standard `unittest` module. We run the tests using `pytest` (**?**) and this comes with many advantages in that all the tests run in the same process

[1] https://github.com/lsst

[2] Line counts include comments but not blank lines. Python interfaces are implemented using `pybind11` and that is counted as C++ code. For the purposes of this count Science pipelines software is defined as the `lsst_distrib` metapackage and does not include code from third party packages.

and requiring global parameters to be well understood, test can be run in parallel in multiple processes, plugins can be enabled to extend testing, and a test report can be created giving details of run times and test failures. Coding standards compliance with PEP 8 (**?**) and code coverage are enabled using `pytest` plugins, and we also check for leaked file descriptors during tests.



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.

*Facility:* LSST