TMS – Add cover note with Versions

Software Management Plan for Lasair

What software will you develop?

* We will develop a broker , called Lasair, which ingests high-bandwidth streams of transient astronomical observations and filters out the interesting substreams that can advance science, as well as adding useful additional information.
* There is an advanced prototype (lasair.lsst.ac.uk) that serves the ZTF survey, and a new version will serve the LSST survey when it starts in 2025.
* Lasair is composed of a set of ingestion modules, coordinated via a Python script, with web pages, web services and Kafka services. Lasair is built with the following high-level components:
  + web/kafka server,
  + ingestion pipeline,
  + database,
  + context classifier.
* These services are built from third-party tools including: Apache, Django, Kafka, , MariaDB, Galera, Cassandra , plotly, jQuery A fuller list is maintained by the project team.

Who are the intended users of your software?

* The Lasair system is open to the public, but we focus on professional astronomers in the UK.
* Users wish to extract what their science wants from a wide and diverse stream of astronomical transients.
* There are two levels of authentication: public and free, self-signup. The second requires only a valid email address.
* We have built a "citizen science" project to identify super-luminous supernovae.

How will you make your software available to your users?

* The Lasair software is available open source from github repositories [https://github.com/lsstuk/lasair-lsst and lasair4], if anyone would like to replicate the system.
* However, we do not expect many users to run the software themselves; rather they will access an instance of the software that we will run on their behalf.
* There is a client library that enable users to interact with the Lasair API.
* The system presents as a website at which filters and queries can be built by users; these can be executed in "pull" mode where a user (or API) makes a request, or where interesting observations are "pushed" to the user as soon as possible.

How will you support those who use your software?

* We have built documentation in many forms: quickstart, how-to as text and video, FAQ, etc.
* Channels available to users are : email to Lasair helpdesk, and the Rubin Community forum. The preferred route is the latter.
* Those who have accounts with Lasair will get occasional emails about upgrades / new releases.

How will your software contribute to research?

* The Lasair system will enable astronomers to get what they want from a billion dollar science survey (LSST), by filtering and storing the firehose of data.
* Specifically, Lasair will improve understanding of supernovae, active galaxies, stellar outbursts, planet formation, and solar system objects.

How will your software relate to other research objects?

* Lasair is useful only because it consumes an astronomical survey (ZTF, LSST), which is itself a large software infrastructure combined with a large telescope.
* Lasair was originally described in Smith, Williams, et. al, https://iopscience.iop.org/article/10.3847/2515-5172/ab020f
* The software is described in https://lasair.readthedocs.io/.
* How will you measure your software's contribution to research?
* The Readthedocs at its front page asks users “If you make use of this, please cite our paper:…”
* The web page also asks users “If you use this service for science, please use the Acknowledgment at the bottom of this page.”

Where will you deposit your software to guarantee its long-term availability?

* The scripts that can configure and deploy an instance of Lasair, are held on the third party GitHub service, which we expect will persist for the long term, due to the volume of important software that it hosts.

RDO-41 “Manual for In-kind Contributors and Recipients” Coding guidelines

RDO-41 states that at minimum, and in the absence of further local coding guidelines, software developed as a Rubin LSST in-kind contribution should do the following:

1. Be designed in collaboration with the recipients, following the recipients’ standard practices.
   1. Use cases, algorithm choices, interfaces (to other packages and target datasets), and (as needed) code structure should all be discussed using the recipients’ communication channels (e.g. LSSTC Slack, recipient Jira or GitHub, working group meetings, etc).

User meetings see wiki pages for the previous meetings.

1. Be developed collaboratively, in a shared version-controlled repository accessible to the recipient group, such that the recipients can follow progress, comment and query, and make contributions of their own.
   1. Commits should represent atomic (i.e. small and indivisible) changes in functionality.
   2. The code should compile, and its tests should pass, before the change is committed.
   3. The commit message should be an informative summary of the change.
   4. Pull requests should be limited to a single feature, and code review practices should be agreed in advance with the recipients (e.g. to prevent reviewer overload)
   5. New contributors should be encouraged and assisted, as a way of ensuring the maintenance and reusability of the code.
   6. Ideally the repository should be open (i.e. publicly visible or available to the Rubin Community) or, as a minimum accessible to the recipient group(s) during package development. One of these must be agreed on with the recipient(s) and noted in the contribution work plan.

See previous question above repo <https://github.com/lsstuk/lasair-lsst and lasair4>

See /tests for the various tests.

PR request process see wiki page

New contributors are due soon when Adler starts

1. Be packaged using common, easily used tools (such as setuptools and pip/pypi in Python, Autotools and cmake in C++, etc).

For Lasair itself this is not the case but

The API is packaged using setuptools/PyPi.

Lasair-examples is a Github repo with example code in to for Lasair users.

1. All functions, packages, dependencies and datasets of or related to the software and necessary for it to function, should also be easily available to the community without licensing or other restriction.

Yes, see principles of operation for software packages.

1. Adhere to reasonable standards, such as those already adopted by the recipients’ related packages, or PEP 8 (Python) and LSSTDM (C++), with style optimized for readability.

No specific standard is currently followed since a small team but there are comments, doc string and principles of operation and unit tests. No problems though have been encountered as result of the lack of a specific style guide.

1. Include a test suite that uses common testing tools, and which is either used by a continuous integration service (like GitHub or travis-CI) or could be used by one.

Test suite is a ctivated usinh Jenkins CI. Coverage extention is an ambition.

1. Be demonstrated and validated following the recipients’ recommendations, preferably via reusable notebooks or scripts that are checked into the package repository.

Collab notebooks, readthedocs, Lasair-examples are available to users.

1. Be fully documented at all times, such that at any point in time the package can be picked up and contributed to by any skilled developer in the recipient group. In addition, a simpler User Guide must be provided for those not accessing the software for development but as end users.
   1. The set of function, class and module docstrings should contain everything the user needs to understand in order to use the code. The set of in-line comments should provide a complete explanation of (and citations to) the algorithms implemented.
   2. The README should enable the recipients to understand what is being developed and how to get involved.
   3. Tutorials should show what the package does, by leading the user through the set of use cases that define the goals of the package. Jupyter notebooks are particularly powerful for teaching Python packages in this way, and can even be used as system tests.
   4. Technical notes (and potentially a journal paper) describe the problem that the package is intended to solve, and what algorithms it implements to solve it.

The user functionality is fully documented in Readthedocs. Skilled user have the API, collab notebooks and Lasair examples. There are also some videos.

Developer documentation is limited in its scope.

1. Be versioned semantically (using e.g. GitHub releases) to improve communication about the code and reproducibility of its outputs.

Versioning is used in the API and the API client code.

The development team do use versioning but are reluctant to use the conventions of semantic versions elsewhere since previous attempts did not offer value and did not improve communication and reproducibility.

1. If appropriate, be published in a suitable journal (such as the Astronomical Journal, which encourages method papers with associated software packages), with the recipients who made significant contributions to the development of the code as co-authors, in accordance with the recipients’ publication policy. The main developers are responsible for publishing the code, but can optionally delegate the paper writing to a recipient collaborator(s). In this case the developers should be credited as early authors on the paper based on the code.

Lasair was originally described in Smith, Williams, et. al, https://iopscience.iop.org/article/10.3847/2515-5172/ab020f

but it would benefit from a follow-on.

1. Be released publicly no later than the time of its first application in a journal paper, unless the recipients have scientific reasons for postponing and successfully petition the Rubin Operations Director (via their IPC) for an exception. (The Rubin Operations Director will take the CEC’s advice on the requested exception, and may delegate their ruling to the Rubin data policy committee or publication board.)

DONE

1. Carry a permissive open source license such as BSD-3-Clause. This is to maximize the re-useability of in-kind generated code by the LSST Science Community (and has the added benefit of high visibility for the code’s authors).

Apache.