DIALS data Documentation

Release 1.0.0

Markus Gerstel

Feb 18, 2019

Contents:

CHAPTER 1

DIALS Regression Data Manager

If you want to know more about what `dials_data` is you can have a read through the background information.

For everything else the main documentation is probably the best start.

1.1 Installation

To install this package in a normal Python environment, run:

```
install dials_data
```

and then you can use it with:

```
dials.data
```

To install `dials_data` in a DIALS installation you need to run:

```
libtbx.pip install dials_data
```

followed by a run of either `libtbx.configure` or `make reconf` to get access to the command line tool.

For more details please take a look at the installation and usage page.

CHAPTER 2

What is dials_data

dials_data is a lightweight, simple python(-only) package. It is used to provide access to data files used in regression tests, but does not contain any of those data files itself.

Although we envisage it mostly being used in a cctbx/DIALS environment for tests in DIALS, dxtbx, xia2 and related packages, it has no dependencies on either cctbx or DIALS, in fact all dependencies are explicitly declared in the setup.py file and are installable via standard setuptools/pip methods. This means dials_data can easily be used in other projects accessing the same data, and can be used in temporary environments such as Travis containers.

2.1 But - why?

In the past DIALS tests used an internal SVN repository called dials_regression to collect any file that would be useful in tests, but should not be distributed with a standard DIALS distribution. This made it easy to add files, from a single JSON file to example files from different detectors to whole datasets. Similarly, all a developer had to do to update the collection was to update the checked out SVN repository.

Over time the downsides of the SVN repository approach became obvious: a checked out copy requires twice the storage space on the local disk. Sparse checkouts are possible, but become increasingly complicated as more files are added. This quickly becomes impractical in distributed testing environments. The disk space required for checkouts can be reduced by compressing the data, but then they need to be unpacked for using the data in tests. By its nature the internal SVN repository was not publically accessible. The data files were too large to convert the repository to a git repository to be hosted on Github, and in any case a git repository was not the best place either to store large amounts of data, as old versions of the data or retired datasets are kept forever, and sparse checkouts would be even more complex. Git LFS would just raise the complexity even further and would incur associated costs. A solution outside SVN/git was built with xia2_regression, which provided a command to download a copy of datasets from a regular webhost. This worked well for a while but still made it complicated to use the data files in tests, as they had to be downloaded – in full – first.

With dxtbx, dials and xia2 moving to pytest we extended the xia2_regression concept into the regression_data fixture to provide a simple way to access the datasets in tests, but the data still needed downloading separately and coult not easily be used outside of the dials repository and not at all outside of a dials distribution. Adding data files was still a very involved process.

dials_data is the next iteration of our solution to this problem.

2.2 What can `dials_data` do

The entire pipeline, from adding new data files, to the automatic download, storage, preparation, verification and provisioning of the files to tests happens in a single, independent Python package.

Data files are versioned without being kept in an SVN or git repository. The integrity of data files can be guaranteed. Files are downloaded/updated as and when required. The provenance of the files is documented, so it can be easily identified who the author of the files is and under what license they have been made available. New datasets can be created, existing ones can be updated easily by anyone using Github pull requests.

CHAPTER 3

Installation and Usage

There are a number of possible ways to install `dials_data`. From the simplest to the most involved these are:

3.1 As an end user

Quite simply: You do not need to install `dials_data`. It does not add any functionality to end users.

3.2 As a developer to run tests with `dials_data`

You may want to install `dials_data` so that you can run regression tests locally. Or you might say that continuous integration should take care of this. Both are valid opinions.

However you do not need to install `dials_data` from source. You can simply run:

```
pip install -U dials_data
```

or, in a cctbx/DIALS environment:

```
libtbx.pip install -U dials_data
```

This will install or update an existing installation of `dials_data`.

In a cctbx/DIALS environment you may have to do a round of `libtbx.configure` or `make reconf` to enable the `dials_data` command line utilities. In a normal Python environment this is not required.

You can then run your tests as usual using:

```
pytest
```

although, depending on the configuration of the code under test, you probably need to run it as:

```
pytest --regression
```

to actually enable those tests depending on files from `dials_data`.

3.3 As a developer to write tests with `dials_data`

Install `dials_data` using `pip` as above.

If your test is written in pytest and you use the fixture provided by `dials_data` then you can use regression datasets in your test by adding the `dials_data` fixture to your test, ie:

```
def test_accessing_a_dataset(dials_data):
    location = dials_data("x4wide")
```

The fixture/variable `dials_data` in the test is a `dials_data.download.DataFetcher` instance, which can be called with the name of the dataset you want to access (here: `x4wide`). If the files are not present on the machine then they will be downloaded. If either the download fails or `--regression` is not specified then the test is skipped.

The return value (`location`) is a `py.path.local` object pointing to the directory containing the requested dataset.

You can see a list of all available datasets by running:

```
dials.data list
```

or by going through the dataset definition files in the repository.

For a new project you may have to set up the `dials_data` fixture first. Assuming you are using `pytest` then simply add the following to a file named `conftest.py` in the top level of your project:

```
import pytest
try:
    from dials_data import *
except ImportError:
    @pytest.fixture
    def dials_data():
        pytest.skip("Test requires python package dials_data")
```

3.4 As a developer who wants to add files to `dials_data`

Follow the steps in *Contributing* to install `dials_data` into a virtual environment.

You can install `dials_data` using `pip` as above *unless* you want to immediately use your dataset definition in tests without waiting for your pull request to be accepted. In this case you can follow the instructions in the next step.

3.5 As a developer who wants to extend `dials_data`

Have a look at the *Contributing* page.

Install your own fork of `dials_data` by running:

```
pip install -e path/to/fork
```

in a cctbx/DIALS environment use `libtbx.pip` respectively, followed by a round of `libtbx.configure` or `make reconf`.

If you made substantial changes or updated your source copy you may also have to run:

```
python setup.py develop
```

or in a cctbx/DIALS environment:

```
libtbx.python setup.py develop
```

followed by a round of `libtbx.configure` or `make reconf`. This will update your python package index and install/update any `dials_data` dependencies if necessary.

To switch back from using your checked out version to the 'official' version of `dials_data` you can uninstall it with:

```
pip uninstall dials_data # or
libtbx.pip uninstall dials_data
```

and then reinstall it following the *instructions at the top of this page*.

3.6 Where are the regression datasets stored?

In order of evaluation:

- If the environment variable `DIALS_DATA` is set and exists or can be created then use that location.
- If the file path `/dls/science/groups/scisoft/DIALS/dials_data` exists and is readable then use this location. This is a shared directory specific to Diamond Light Source.
- If the environment variable `LIBTBX_BUILD` is set and the directory `dials_data` exists or can be created underneath that location then use that.
- Use `~/.cache/dials_data` if it exists or can be created.
- Otherwise `dials_data` will fail with a RuntimeError.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/dials/data/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Add Datasets

DIALS data was planned to support a more or less arbitrary number of datasets. You can contribute by adding more.

4.1.3 Write Documentation

DIALS Regression Data could always use more documentation, whether as part of the official DIALS Regression Data docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.4 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dials/data/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *dials_data* for local development.

1. Fork the *dials_data* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dials_data.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dials_data
$ cd dials_data/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 dials_data tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests, unless you are adding or updating a dataset.

2. If you add or update a dataset then make individual pull requests for each dataset, so that they can be discussed and approved separately.
3. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in HISTORY.rst.
4. The pull request should work for all supported Python versions. Check https://travis-ci.com/dials/data/pull_requests

4.4 Deploying

Any commit on the master branch is now automatically deployed to PyPI, so there is no need to play around with tags or version numbers on a regular basis.

For slightly larger changes make sure that the entry in HISTORY.rst is updated, and then run:

```
$ bumpversion minor # possible: major / minor, do not use patch
```

Travis will then automatically tag the commit once it hits the master branch and the tests pass, and then deploy to PyPI as usual.

CHAPTER 5

History

5.1 1.0 (2019-02-16)

- Add functions for forward-compatibility
- Enable new release process including automatic deployment of updates

5.2 0.6 (2019-02-15)

- Added datasets blend_tutorial, thaumatin_i04

5.3 0.5 (2019-01-24)

- Added documentation
- Added datasets fumarase, vmxi_thaumatin

5.4 0.4 (2018-01-11)

- Beta release
- Added datasets insulin, pychef
- Automated generation of hashinfo files via Travis

5.5 0.3 (2019-01-09)

- Dataset download mechanism

- Added dataset x4wide

5.6 0.2 (2019-01-08)

- Alpha release
- Basic command line interface
- pytest fixture

5.7 0.1 (2018-11-02)

- First automatic deployment and release on PyPI