OpenSCM-Zenodo Documentation

Release 0.1.3

Zebedee Nicholls, Jared Lewis

Jan 19, 2022

DOCUMENTATION

OpenSCM-Zenodo provides a basic command-line interface for creating new versions of Zenodo uploads and for uploading files to Zenodo.

CHAPTER

ONE

LICENSE

OpenSCM-Zenodo is free software under a BSD 3-Clause License, see LICENSE.

1.1 Installation

OpenSCM-Zenodo can be installed with pip

```
pip install openscm-zenodo
```

1.2 Usage

The typical usage workflow looks something like the following:

1. Create an API token for either zenodo.org or sandbox.zenodo.org. Make sure you that you specify the `ZENODO_URL` which matches the account for which you created the token.
2. Setup your metadata in a `.json` file, this is your `DEPOSIT_METADATA`
3. Create a new version for your record i.e. `ZENODO_TOKEN=TOKEN openscm-zenodo create-new-version DEPOSITION_ID DEPOSIT_METADATA --zenodo-url ZENODO_URL`. Note the new version which is output by this program e.g. `739845`
4. Get the bucket (in which you'll upload files) for your new version i.e. `ZENODO_TOKEN=TOKEN openscm-zenodo get-bucket VERSION --zenodo-url ZENODO_URL`. Note the bucket which is output by this program e.g. `e428bd2f-84e5-49ae-84ed-f42da1b8e0da`.
5. Upload your file to the bucket e.g. `openscm-zenodo upload FILE_TO_UPLOAD BUCKET --zenodo-url ZENODO_URL` (or specify the root directory that you want to strip from the full filepath e.g. `openscm-zenodo upload FILE_TO_UPLOAD BUCKET --root-dir /path/to/strip --zenodo-url ZENODO_URL`)
6. Go to zenodo and check your new record, the publishing step is performed manually via the Zenodo GUI

1.3 Development

If you're interested in contributing to OpenSCM-Zenodo, we'd love to have you on board! This section of the docs will (once we've written it) detail how to get setup to contribute and how best to communicate.

1.3.1 Contributing

All contributions are welcome, some possible suggestions include:

- tutorials (or support questions which, once solved, result in a new tutorial :D)
- blog posts
- improving the documentation
- bug reports
- feature requests
- pull requests

Please report issues or discuss feature requests in the OpenSCM-Zenodo issue tracker. If your issue is a feature request or a bug, please use the templates available, otherwise, simply open a normal issue :)

As a contributor, please follow a couple of conventions:

- Create issues in the OpenSCM-Zenodo issue tracker for changes and enhancements, this ensures that everyone in the community has a chance to comment
- Be welcoming to newcomers and encourage diverse new contributors from all backgrounds: see the Python Community Code of Conduct
- Only push to your own branches, this allows people to force push to their own branches as they need without fear or causing others headaches
- Start all pull requests as draft pull requests and only mark them as ready for review once they've been rebased onto master, this makes it much simpler for reviewers
- Try and make lots of small pull requests, this makes it easier for reviewers and faster for everyone as review time grows exponentially with the number of lines in a pull request

1.3.2 Getting setup

To get setup as a developer, we recommend the following steps (if any of these tools are unfamiliar, please see the resources we recommend in *Development tools*):

1. Install conda and make
2. Run `make virtual-environment`, if that fails you can try doing it manually
    1. Change your current directory to OpenSCM-Zenodo's root directory (i.e. the one which contains `README.rst`), `cd openscm-zenodo`
    2. Create a virtual environment to use with OpenSCM-Zenodo `python3 -m venv venv`
    3. Activate your virtual environment `source ./venv/bin/activate`
    4. Upgrade pip `pip intall --upgrade pip`
    5. Install the development dependencies (very important, make sure your virtual environment is active before doing this) `pip install -e .[dev]`
3. Make sure the tests pass by running `make test-all`, if that fails the commands are
    1. Activate your virtual environment `source ./venv/bin/activate`
    2. Run the unit and integration tests `pytest --cov -r a --cov-report term-missing`
    3. Test the notebooks `pytest -r a --nbval ./notebooks --sanitize ./notebooks/tests_sanitize.cfg`

Getting help

Whilst developing, unexpected things can go wrong (that's why it's called 'developing', if we knew what we were doing, it would already be 'developed'). Normally, the fastest way to solve an issue is to contact us via the issue tracker. The other option is to debug yourself. For this purpose, we provide a list of the tools we use during our development as starting points for your search to find what has gone wrong.

Development tools

This list of development tools is what we rely on to develop OpenSCM-Zenodo reliably and reproducibly. It gives you a few starting points in case things do go inexplicably wrong and you want to work out why. We include links with each of these tools to starting points that we think are useful, in case you want to learn more.

- Git
- Make
- Conda virtual environments
- Pip and pip virtual environments
- Tests
    - we use a blend of pytest and the inbuilt Python testing capabilities for our tests so checkout what we've already done in `tests` to get a feel for how it works
- Continuous integration (CI) (also brief intro blog post and a longer read)
    - we use GitHub CI for our CI but there are a number of good providers
- Jupyter Notebooks
    - Jupyter is automatically included in your virtual environment if you follow our *Getting setup* instructions

- Sphinx

Other tools

We also use some other tools which aren't necessarily the most familiar. Here we provide a list of these along with useful resources.

- Regular expressions
  - we use regex101.com to help us write and check our regular expressions, make sure the language is set to Python to make your life easy!

1.3.3 Formatting

To help us focus on what the code does, not how it looks, we use a couple of automatic formatting tools. These automatically format the code for us and tell use where the errors are. To use them, after setting yourself up (see *Getting setup*), simply run `make format` (and `make format-notebooks` to format notebook code). Note that `make format` can only be run if you have committed all your work i.e. your working directory is 'clean'. This restriction is made to ensure that you don't format code without being able to undo it, just in case something goes wrong.

1.3.4 Buiding the docs

After setting yourself up (see *Getting setup*), building the docs is as simple as running `make docs` (note, run `make -B docs` to force the docs to rebuild and ignore make when it says '... index.html is up to date'). This will build the docs for you. You can preview them by opening `docs/build/html/index.html` in a browser.

For documentation we use Sphinx. To get ourselves started with Sphinx, we started with this example then used Sphinx's getting started guide.

Gotchas

To get Sphinx to generate pdfs (rarely worth the hassle), you require Latexmk. On a Mac this can be installed with `sudo tlmgr install latexmk`. You will most likely also need to install some other packages (if you don't have the full distribution). You can check which package contains any missing files with `tlmgr search --global --file [filename]`. You can then install the packages with `sudo tlmgr install [package]`.

Docstring style

For our docstrings we use numpy style docstrings. For more information on these, here is the full guide and the quick reference we also use.

1.3.5 Releasing

1. Test installation with dependencies `make test-install`
2. Update `CHANGELOG.rst`
   - add a header for the new version between `master` and the latest bullet point
   - this should leave the section underneath the master header empty
3. `git add .`

4. `git commit -m "Prepare for release of vX.Y.Z"`
5. `git tag vX.Y.Z`
6. Test version updated as intended with `make test-install`
7. Push to GitHub with `git push && git push --tags`, GitHub actions do the rest

1.3.6 Why is there a `Makefile` in a pure Python repository?

Whilst it may not be standard practice, a `Makefile` is a simple way to automate general setup (environment setup in particular). Hence we have one here which basically acts as a notes file for how to do all those little jobs which we often forget e.g. setting up environments, running tests (and making sure we're in the right environment), building docs, setting up auxillary bits and pieces.

1.4 Command-line interface

1.4.1 openscm-zenodo

OpenSCM-Zenodo's command-line interface

```
openscm-zenodo [OPTIONS] COMMAND [ARGS]...
```

Options

--log-level <log_level>

> **Options** DEBUG | INFO | WARNING | ERROR | EXCEPTION | CRITICAL

create-new-version

Create new version of a Zenodo record (i.e. a specific Zenodo deposition ID).

`deposition_id` is the deposition ID of any existing version DOI, but crucially **not** the DOI which represents all versions of the record (this won't work). The printed value is the deposition ID of the DOI which represents the new, specific version of the record.

The deposit metadata should be a `.json` file. It will be read and validated before the new version is created.

```
openscm-zenodo create-new-version [OPTIONS] DEPOSITION_ID DEPOSIT_METADATA
```

Options

--zenodo-url <zenodo_url>
Zenodo server to which to upload.

> **Default** sandbox.zenodo.org
>
> **Options** sandbox.zenodo.org | zenodo.org

--token <token>

Arguments

DEPOSITION_ID
Required argument

DEPOSIT_METADATA
Required argument

Environment variables

ZENODO_TOKEN

Provide a default for *--token*

get-bucket

Get the bucket associated with a given Zenodo deposition ID (`deposition_id`)

The `upload` command can then be used to upload files to this bucket.

This command is handy when you know your deposition ID but you've forgotten the bucket address.

```
openscm-zenodo get-bucket [OPTIONS] DEPOSITION_ID
```

Options

--zenodo-url <zenodo_url>
Zenodo server to which to upload.

Default sandbox.zenodo.org

Options sandbox.zenodo.org | zenodo.org

--token <token>

Arguments

DEPOSITION_ID
Required argument

Environment variables

ZENODO_TOKEN

Provide a default for *--token*

upload

Upload a file to a Zenodo bucket

`file_to_upload` will be uploaded to the Zenodo bucket specified by `bucket`.

```
openscm-zenodo upload [OPTIONS] FILE_TO_UPLOAD BUCKET
```

Options

--root-dir <root_dir>
Root directory (removed from file paths before uploading)

--zenodo-url <zenodo_url>
Zenodo server to which to upload.

Default sandbox.zenodo.org

Options sandbox.zenodo.org | zenodo.org

--token <token>

Arguments

FILE_TO_UPLOAD
Required argument

BUCKET
Required argument

Environment variables

ZENODO_TOKEN

Provide a default for *--token*

1.5 Uploading API

File uploading handling

Thanks https://gist.github.com/tyhoff/b757e6af83c1fd2b7b83057adf02c139 for the progress bar.

openscm_zenodo.uploading.**upload_with_progress_bar**(*filepath*, *upload_url*)
Upload file with a progress bar

Parameters

- **filepath** (*str*) – Path to file to upload
- **upload_url** (*str*) – URL to put the file onto

1.6 Zenodo API

Zenodo interactions handling

opensmc_zenodo.zenodo.**create_new_zenodo_version**(*deposition_id*, *zenodo_url*, *token*, *deposit_metadata*)
Create a new version of a Zenodo record (i.e. a specific deposition ID)

Parameters

- **deposition_id** (*str*) – Deposition ID of any DOI which represents a specific version of the record, but crucially **not** the DOI which represents all versions of the record (this won't work).
- **zenodo_url** (*str*) – Zenodo url to upload the file to (e.g. `sandbox.zenodo.org` or `zenodo.org`)
- **token** (*str*) – Token to use to authenticate the request
- **deposit_metadata** (*str*) – Path to file containing metadata for this new version

Returns The deposition ID of the new version of the record

Return type str

opensmc_zenodo.zenodo.**get_bucket_id**(*deposition_id*, *zenodo_url*, *token*)
Get bucket ID for a given Zenodo deposition ID

Parameters

- **deposition_id** (*str*) – Deposition ID to check
- **zenodo_url** (*str*) – Zenodo url to upload the file to (e.g. `sandbox.zenodo.org` or `zenodo.org`)
- **token** (*str*) – Token to use to authenticate the request

Returns The bucket associated with `deposition_id`

Return type str

opensmc_zenodo.zenodo.**upload_file**(*filepath*, *bucket*, *zenodo_url*, *token*, *root_dir=None*)
Upload file to Zenodo

Parameters

- **filepath** (*str*) – Path to file to upload
- **bucket** (*str*) – Bucket to upload the file to
- **zenodo_url** (*str*) – Zenodo url to upload the file to (e.g. `sandbox.zenodo.org` or `zenodo.org`)
- **token** (*str*) – Token to use to authenticate the upload
- **root_dir** (*str*) – Path to remove from filename before uploading. If not supplied, only the filename is used for uploading i.e. all preceeding directories are stripped.

1.7 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

The changes listed in this file are categorised as follows:

- Added: new features
- Changed: changes in existing functionality
- Deprecated: soon-to-be removed features
- Removed: now removed features
- Fixed: any bug fixes
- Security: in case of vulnerabilities.

1.7.1 v0.1.3 - 2022-01-20

Added

- (#5) Ability to specify the root directory to be removed when uploading files

1.7.2 v0.1.2 - 2021-08-06

Fixed

- (#4) Handling of uploads when the Zenodo record does not have any published versions (hence no "latest" information)

1.7.3 v0.1.1 - 2021-03-10

Added

- (#3) Clearer error message if metadata upload fails during creation of a new version

1.7.4 v0.1.0 - 2021-03-03

Added

- (#2) Add basic docs
- (#1) Add basic repository setup and functionality

CHAPTER

TWO

INDEX

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Symbols

B

C

D

F

G

M

O

U