invenio-records-files Documentation

Release 1.2.1

CERN

Mar 20, 2020

Contents

Invenio module that provides basic API for integrating Invenio-Records and Invenio-Files-REST.

Features:

- Records creation
- Files creation
- Accessing files
- Files metadata management
- Files extraction from records

Further documentation is available on https://invenio-records-files.readthedocs.io/en/latest/usage.html#initializations

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-Records-Files.

1.1 Installation

Invenio-Records-Files is on PyPI so all you need is:

```
$ pip install invenio-records-files
```

1.2 Configuration

Invenio-Records-Files configuration.

invenio_records_files.config.**RECORDS_FILES_REST_ENDPOINTS = {}**

REST endpoints configuration.

You can configure the REST API endpoint to access the record's files as follows:

```
RECORDS_FILES_REST_ENDPOINTS = {
    '<*_REST_ENDPOINTS>': {
        '<endpoint-prefix>': '<endpoint-suffix>',
    }
}
```

- <*_REST_ENDPOINTS> corresponds to Invenio-Records-REST endpoint configurations names that you have defined in your application.
- <endpoint-prefix> is the unique name of the endpoint configuration as it is defined in Invenio-Records-REST like configuration. This needs to match an already existing endpoint name in the <*_REST_ENDPOINTS> configuration.
- <endpoint-suffix> is the endpoint path name to access the record's files.

```
{'recid': '/myawesomefiles'} -> /records/1/myawesomefiles
```

An example of this configuration is provided in the Integration with Invenio REST API section of the documentation.

1.3 Usage

Integration of records and files for Invenio.

Invenio-Records-Files provides basic API for integrating Invenio-Records and Invenio-Files-REST.

1.3.1 Initialization

First create a Flask application:

```
>>> from flask import Flask
>>> app = Flask('myapp')
>>> app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite://'
```

Records-Files has no Flask extension, however it is dependent on Invenio-Records and Invenio-Files-REST which must be initialized first:

```
>>> from invenio_db import InvenioDB
>>> ext_db = InvenioDB(app)
>>> from invenio_records import InvenioRecords
>>> from invenio_files_rest import InvenioFilesREST
>>> ext_filesrest = InvenioFilesREST(app)
>>> ext_records = InvenioRecords(app)
```

In order for the following examples to work, you need to work within a Flask application context so let's push one:

```
>>> ctx = app.app_context()
>>> ctx.push()
```

Also, for the examples to work you need to create the database and tables (note, in this example you use an in-memory SQLite database):

```
>>> from invenio_db import db
>>> db.create_all()
```

Lastly, since you're managing files, you need to create a default location. Here you will create a location in a temporary directory:

```
>>> import tempfile
>>> tmppath = tempfile.mkdtemp()
>>> from invenio_files_rest.models import Location
>>> db.session.add(Location(name='default', uri=tmppath, default=True))
>>> db.session.commit()
```

1.3.2 Creating a record

Import Invenio-Records-Files basic API *invenio_records_files.api.Record*:

```
>>> from invenio_records_files.api import Record
```

This *Record* class has a special property `files` through which you can access and create files. By default the class creates a bucket when you create a record:

```
>>> record = Record.create({})
>>> len(record.files)
0
```

You can also just create a record without an associated bucket:

```
>>> record_nobucket = Record.create({}, with_bucket=False)
>>> record_nobucket.files is None
True
```

1.3.3 Creating files

You are now ready to create you first file using the Invenio-Records-Files API:

```
>>> from six import BytesIO
>>> record.files['hello.txt'] = BytesIO(b'Hello, World')
```

In the above example you created a file named `hello.txt` as a new object in the record bucket.

1.3.4 Accessing files

You can access the above file through the same API:

```
>>> len(record.files)
1
>>> 'hello.txt' in record.files
True
>>> fileobj = record.files['hello.txt']
>>> print(fileobj.key)
hello.txt
```

1.3.5 Metadata for files

Besides creating files you can also assign metadata to files:

```
>>> fileobj['filetype'] = 'txt'
>>> print(record.files['hello.txt']['filetype'])
txt
```

Certain key names are however reserved:

```
>>> fileobj['key'] = 'test'
Traceback (most recent call last):
  ...
KeyError: 'key'
```

The reserved key names are all the properties which already exist in `invenio_files_rest.models.ObjectVersion`.

You can however still use the reserved keys for getting metadata:

```
>>> print(fileobj['key'])
hello.txt
```

1.3.6 Dumping files

You can make a dictionary of all files:

```
>>> dump = record.files.dumps()
>>> for k in sorted(dump[0].keys()):
...     print(k)
bucket
checksum
file_id
filetype
key
size
version_id
```

1.3.7 Retrieve files from a record

Invenio-Records-Files provides an utility to retrieve files of a given record.

```
>>> from invenio_records_files.utils import record_file_factory
>>> fileobj = record_file_factory(None, record, 'hello.txt')
>>> print(fileobj.key)
hello.txt
```

If the file does not exist or the record class has no files property, the factory will return `None`:

```
>>> fileobj = record_file_factory(None, record, 'invalid')
>>> fileobj is None
True
```

Some other Invenio modules such as Invenio-Previewer already uses it to programmatically access record's files.

1.3.8 Integration with Invenio REST API

Invenio-Records-Files provides REST endpoints to retrieve or upload the files of a record:

```
# Upload a file named example.txt to the record with pid of 1
$ curl -X PUT http://localhost:5000/api/records/1/files/example.txt \
       -H "Content-Type: application/octet-stream" \
       --data-binary @example.txt

# Get the list of files for this record
$ curl -X GET http://localhost:5000/api/records/1/files/

# Download the file named ``example.txt`` of this record
```

(continues on next page)

(continued from previous page)

```
$ curl -X GET http://localhost:5000/api/records/1/files/example.txt \
       -o example.txt
```

Invenio-Records-Files provides the same REST endpoints for bucket and objects available in Invenio-Files-REST, by implicitly injecting the record's bucket ID to the request.

For example given the following configuration:

```
# Invenio-Records-REST
RECORDS_REST_ENDPOINTS = {
    recid: {
        # ...,
        item_route='/records/<pid(recid):pid_value>',
        #...,
    },
    docid: {
        # ...,
        item_route='/documents/<pid(docid):pid_value>',
        #...,
    }
}
# Invenio-Records-Files
RECORDS_FILES_REST_ENDPOINTS = {
    'RECORDS_REST_ENDPOINTS': {
        'recid': '/files',
        'docid': '/doc-files',
    },
    'DEPOSIT_REST_ENDPOINTS': {
        'depid': '/deposit-files,
    }
}
```

You can access the files of a record with PID `1` using the URL `/api/records/1/files` or of a document with PID `123` using the URL `/api/documents/123/doc-files`.

You can access a specific file, for instance `example.txt`, with the following URL `/api/records/1/files/example.txt`.

Invenio-Records-Files endpoint offers the same functionality provided by Invenio-Files-REST API. More information about handling files through the REST API can be found here.

1.3.9 Integration with Invenio-Records-UI

If you are using Invenio-Records-UI, you can easily add new views by defining new endpoints into your `RECORDS_UI_ENDPOINTS` configuration. In particular, you can add the `file_download_ui` endpoint:

```
RECORDS_UI_ENDPOINTS = dict(
    recid=dict(
        # ...
        route='/records/<pid_value/files/<filename>',
        view_imp='invenio_records_files.utils:file_download_ui',
        record_class='invenio_records_files.api:Record',
    )
)
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

2.1.1 Record API

API for manipulating files associated to a record.

class invenio_records_files.api.**FileObject**(*obj*, *data*)
Wrapper for files.

Bind to current bucket.

dumps()
Create a dump of the metadata associated to the record.

get(*key*, *default=None*)
Proxy to `obj`.

Parameters **key** – Metadata key which holds the value.

Returns Metadata value of the specified key or default.

get_version(*version_id=None*)
Return specific version `ObjectVersion` instance or HEAD.

Parameters **version_id** – Version ID of the object.

Returns `ObjectVersion` instance or HEAD of the stored object.

class invenio_records_files.api.**FilesIterator**(*record*, *bucket=None*, *file_cls=None*)
Iterator for files.

Initialize iterator.

dumps(*bucket=None*)
Serialize files from a bucket.

Parameters **bucket** – Instance of files `invenio_files_rest.models.Bucket`. (Default: `self.bucket`)

Returns List of serialized files.

flush()
Flush changes to record.

keys
Return file keys.

next()
Python 2.7 compatibility.

rename(**args*, ***kwargs*)
Rename a file.

Parameters

- **old_key** – Old key that holds the object.
- **new_key** – New key that will hold the object.

Returns The object that has been renamed.

sort_by(**ids*)
Update files order.

Parameters **ids** – List of ids specifying the final status of the list.

class invenio_records_files.api.**FilesMixin**
Implement files attribute for Record models.

Note: Implement `_create_bucket()` in subclass to allow files property to automatically create a bucket in case no bucket is present.

---

file_cls
File class used to generate the instance of files. Default to *FileObject*

alias of *FileObject*

files
Get files iterator.

Returns Files iterator.

files_iter_cls
Files iterator class used to generate the files iterator. Default to *FilesIterator*

alias of *FilesIterator*

class invenio_records_files.api.**Record**(**args*, ***kwargs*)
Record class with associated bucket.

The record class implements a one-to-one relationship between a bucket and a record. A bucket is automatically created and associated with the record when the record is created with *Record.create()* (unless `with_bucket` is set to `False`).

The bucket id is stored in the record metadata (by default in the `_bucket` key). You can implement your dump/load behavior for storing the bucket id in the record (or possibly somewhere else). You do this by creating a subclass of this class, and overriding the two classmethods *Record.dump_bucket()* and *Record.load_bucket()*.

Initialize the record.

bucket
Get bucket instance.

bucket_id
Get bucket id from record metadata.

classmethod create(*data*, *id_=None*, *with_bucket=True*, ***kwargs*)
Create a record and the associated bucket.

Parameters **with_bucket** – Create a bucket automatically on record creation.

classmethod create_bucket(*data*)
Create a bucket for this record.

Override this method to provide more advanced bucket creation capabilities. This method may return a new or existing bucket, or may return None, in case no bucket should be created.

delete(*force=False*)
Delete a record and also remove the RecordsBuckets if necessary.

Parameters **force** – True to remove also the *RecordsBuckets* object.

Returns Deleted record.

classmethod dump_bucket(*data*, *bucket*)
Dump the bucket id into the record metadata.

Override this method to provide custom behavior for storing the bucket id in the record metadata. By default the bucket id is stored in the `_bucket` key. If you override this method, make sure you also override *Record.load_bucket()*.

This method is called after the bucket is created, but before the record is created in the database.

Parameters

- **data** – A dictionary of the record metadata.
- **bucket** – The created bucket for the record.

classmethod load_bucket(*record*)
Load the bucket id from the record metadata.

Override this method to provide custom behavior for retrieving the bucket id from the record metadata. By default the bucket id is retrieved from the `_bucket` key. If you override this method, make sure you also override *Record.dump_bucket()*.

Parameters **record** – A record instance.

2.1.2 Utilities

Implementation of various utility functions.

invenio_records_files.utils.**file_download_ui**(*pid*, *record*, *_record_file_factory=None*, ***kwargs*)
File download view for a given record.

Plug this method into your `RECORDS_UI_ENDPOINTS` configuration:

```
RECORDS_UI_ENDPOINTS = dict(
    recid=dict(
        # ...
        route='/records/<pid_value>/files/<filename>',
        view_imp='invenio_records_files.utils:file_download_ui',
        record_class='invenio_records_files.api:Record',
    )
)
```

If `download` is passed as a querystring argument, the file is sent as an attachment.

Parameters

- **pid** – The `invenio_pidstore.models.PersistentIdentifier` instance.
- **record** – The record metadata.

`invenio_records_files.utils.`**record_file_factory**(*pid*, *record*, *filename*)

Get file from a record.

Parameters

- **pid** – Not used. It keeps the function signature.
- **record** – Record which contains the files.
- **filename** – Name of the file to be returned.

Returns File object or `None` if not found.

`invenio_records_files.utils.`**sorted_files_from_bucket**(*bucket*, *keys=None*)

Return files from bucket sorted by given keys.

Parameters

- **bucket** – `Bucket` containing the files.
- **keys** – Keys order to be used.

Returns Sorted list of bucket items.

2.1.3 Models

Define relation between records and buckets.

class `invenio_records_files.models.`**RecordsBuckets**(***kwargs*)

Relationship between Records and Buckets.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

bucket

Relationship to the bucket.

bucket_id

Bucket related with the record.

classmethod create(*record*, *bucket*)

Create a new RecordsBuckets and adds it to the session.

Parameters

- **record** – Record used to relate with the `Bucket`.
- **bucket** – Bucket used to relate with the `Record`.

Returns The *RecordsBuckets* object created.

record
It is used by SQLAlchemy for optimistic concurrency control.

record_id
Record related with the bucket.

2.1.4 Links

Link for file bucket creation.

invenio_records_files.links.**default_bucket_link_factory**(*pid*)
Factory for record bucket generation.

invenio_records_files.links.**default_record_files_links_factory**(*pid*, *record=None*, ***kwargs*)
Factory for record files links generation.

Parameters **pid** – A Persistent Identifier instance.

Returns Dictionary containing a list of useful links for the record.

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

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

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-records-files/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.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-Records-Files could always use more documentation, whether as part of the official Invenio-Records-Files docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-records-files/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 :)

3.1.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/invenio-records-files.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 invenio-records-files
$ cd invenio-records-files/
$ pip install -e .[all]
```

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 tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

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

```
$ git add .
$ git commit -s -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.

3.1.3 Pull Request Guidelines

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

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.com/inveniosoftware/invenio-records-files/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.2.1 (released 2019-11-21)

- increase invenio-files-rest version to provide signals for d eletion and uploading files

Version 1.2.0 (released 2019-11-19)

- Adds link factory for files and record
- Fixes the blueprints building

Version 1.1.1 (released 2019-07-31)

- Fixes missing entry point definition for the extension, causing the extension and config not to be loaded.
- Fix issue with when used with Flask-Talisman.

Version 1.1.0 (released 2019-07-29)

- Backward incompatible changes to API.

Version 1.0.0 (released 2019-07-23)

- Initial public release.

3.3 License

Note: In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an Intergovernmental Organization or submit itself to any jurisdiction.

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Dinos Kousidis
- Jacopo Notarstefano
- Javier Delgado
- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Nikos Filippakis
- Sami Hiltunen
- Tibor Simko

Python Module Index

i

Index

S