invenio-files-rest Documentation

Release 1.5.0

CERN

Mar 02, 2023

CONTENTS

Invenio-Files-REST is a files storage module. It allows you to store and retrieve files in a similar way to Amazon S3 APIs.

Features:

- Files storage with configurable storage backends
- Secure REST APIs
- Support for large file uploads and multipart upload.
- Customizable access control
- File integrity monitoring

Further documentation is available on https://invenio-files-rest.readthedocs.io/.

CHAPTER

ONE

USER'S GUIDE

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

1.1 Overview

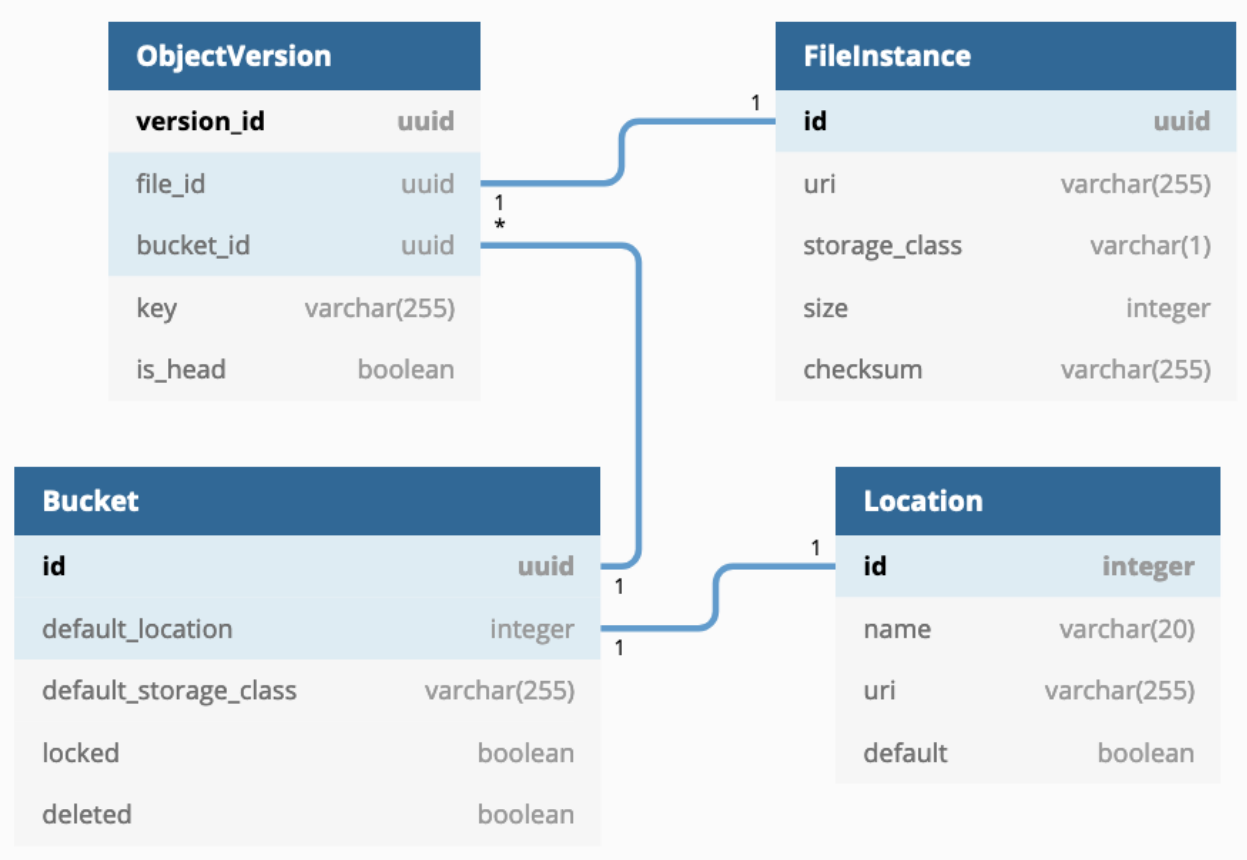
Invenio-Files-REST is a files storage module. It allows you to store and retrieve files in a similar way to Amazon S3 APIs. It provides a set of features:

- An abstraction of physical files storage.
- Configurable storage backends with the ability to build your very own.
- A robust REST API.
- Highly customizable access-control.
- Secure file handling.
- Integrity checking mechanism.
- Support for large file uploads and multipart upload.
- Signals for system events.

The REST API follows best practices and supports, e.g.:

- Content negotiation and links headers.
- Cache control via ETags and Last-Modified headers.
- Optimistic concurrency control via ETags.
- Rate-limiting, Cross-Origin Resource Sharing, and various security headers.

Here is an introduction of the main concepts.



1.1.1 The physical layer

Provides physical access to files defining the storage locations and how to perform operations.

Location

A `Location` is a representation of a storage system. A location is described by its `name` and `URI`. The URI could be a path in a local directory or a remote system. For example, a location could have as a name `shared-folder` and URI `/mnt/shared`.

Among all defined locations, one has to be set as the `default`.

You can learn how to use Locations in the section *Create a location*.

Storage

A `Storage` provides the interface to interact with a `Location` and perform basic operations such as retrieve, store or delete files.

Multiple storage can be useful, for example, to represent offline/online location so that the system known if it can serve files and/or what is the reliability.

By default, Invenio-Files-REST implements a storage for local files *invenio_files_rest.storage.PyFSFileStorage*. It controls how files are physically stored, for example the folders and files structure.

You can create new storage implementations and configure Invenio to use any existing storage. Check *Storage Backends* documentation for detailed instructions on how to build your own.

An example of a remote storage system is implemented in the module Invenio-S3 which offers integration with any S3 REST API compatible object storage.

FileInstance

A file on disk is represented by a `FileInstance`. A FileInstance describes the path to the file, the used storage, the size and the checksum of the file on disk.

To summarize

FileInstances are stored on disk in a specified Location using Storage APIs.

1.1.2 The abstraction layer

Provides an abstract way to logically represent, organise and manipulate files. This abstraction layer allows to perform files operations without physically accessing files.

ObjectVersion

An `ObjectVersion` is the logical representation of a version of a file and metadata at a specific point in time. It contains the reference to the `FileInstance` that it corresponds. For example, the file name is stored in the ObjectVersion metadata.

Note: File names could contain non-alphanumeric characters, which could be a problem depending on your file system. For example, a user could upload a file named `thesis&first*v1.pdf`.

In Invenio-Files-REST, the default `Storage` will save the file in a tree of directories that are uniquely named. The file name will be changed to `data` (with no extension) and the original file name will be stored in the metadata of the ObjectVersion.

The final path to the file on disk will be something like `/mnt/shared/2a/4f/39-5033-af42-k42m/data`.

When an ObjectVersion has no reference to a `FileInstance`, it marks that the file has been logically (and not physically) deleted. This is also known as delete marker (or soft deletion).

Given multiple ObjectVersions of the same file, the latest (or most recent) version is referred to as the `HEAD`.

ObjectVersions are very useful to perform operations on file's metadata without directly accessing to the storage. For example, given that the filename is part of the ObjectVersion metadata, a rename operation is simply a database query to change its value.

Moreover, multiple `ObjectVersion` can reference the same `FileInstance`. This allows to perform some operations more efficiently, such as create a snapshot without physically duplicating files or migrating data.

Let's see an example

A user uploads a new file called `thesis.pdf`.

With location and storage mentioned above, the file will be physically stored in `/mnt/shared` in a tree of folders and with filename `data` (its FileInstance URI will be something like `/<folders>/data`).

The logical representation of the file, the ObjectVersion, will contain the reference to that FileInstance and it will also store the filename `thesis.pdf`.

If, afterwards, the file is renamed to `mythesis.pdf`, a new ObjectVersion will be created with the new filename keeping the reference to the same FileInstance.

If the file is then removed, a new ObjectVersion will be created with no reference to any FileInstance, without physically deleting the file on disk.

Bucket

A `Bucket` is a container for `ObjectVersion` objects. Just as in a traditional file system where files are contained in folders, each `ObjectVersion` has to be contained in a `Bucket`. The `Bucket` has a reference to the `Location` where files are stored.

Buckets are useful to create collections of objects and to act on them. For example, bucket keeps track of the total size of the object if contains and allows definitions of quotas.

A bucket can also be marked as deleted, in which case the contents become inaccessible.

To summarize

Bucket contains ObjectVersions, a version of a file and its metadata. Each ObjectVersion has a reference to a FileInstance.

1.1.3 REST APIs

Invenio-Files-REST provides a set of REST APIs to create or manage resources such as Buckets or ObjectVersions. You can learn more about it in the *REST APIs* section of the documentation.

1.2 Installation

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

```
$ pip install invenio-files-rest
```

1.3 Configuration

Invenio Files Rest module configuration file.

`invenio_files_rest.config.`**`FILES_REST_DEFAULT_MAX_FILE_SIZE = None`**

Default maximum file size for a bucket in bytes. *None* if unlimited.

`invenio_files_rest.config.`**`FILES_REST_DEFAULT_QUOTA_SIZE = None`**

Default quota size for a bucket in bytes. *None* if unlimited.

`invenio_files_rest.config.`**`FILES_REST_DEFAULT_STORAGE_CLASS = 'S'`**

Default storage class. Must be one of *FILES_REST_STORAGE_CLASS_LIST*.

`invenio_files_rest.config.`**`FILES_REST_FILE_TAGS_HEADER = 'X-Invenio-File-Tags'`**

Header for updating file tags.

invenio_files_rest.config.**FILES_REST_FILE_URI_MAX_LEN = 255**

Maximum length of the FileInstance.uri field.

> **Warning:** Setting this variable to anything higher than 255 is only supported with PostgreSQL database.

invenio_files_rest.config.**FILES_REST_MIN_FILE_SIZE = 1**

Minimum file size when uploading, in bytes (do not allow empty files).

invenio_files_rest.config.**FILES_REST_MULTIPART_CHUNKSIZE_MAX = 5368709120**

Maximum chunk size in bytes of multipart objects.

invenio_files_rest.config.**FILES_REST_MULTIPART_CHUNKSIZE_MIN = 5242880**

Minimum chunk size in bytes of multipart objects.

invenio_files_rest.config.**FILES_REST_MULTIPART_EXPIRES = datetime.timedelta(days=4)**

Time delta after which a multipart upload is considered expired.

invenio_files_rest.config.**FILES_REST_MULTIPART_MAX_PARTS = 10000**

Maximum number of parts when uploading files with multipart uploads.

invenio_files_rest.config.**FILES_REST_MULTIPART_PART_FACTORIES = ['invenio_files_rest.views:default_partfactory', 'invenio_files_rest.views:ngfileupload_partfactory']**

Import path of factories used when parsing upload params for multipart.

invenio_files_rest.config.**FILES_REST_OBJECT_KEY_MAX_LEN = 255**

Maximum length of the ObjectVersion.key field.

> **Warning:** Setting this variable to anything higher than 255 is only supported with PostgreSQL database.

invenio_files_rest.config.**FILES_REST_PERMISSION_FACTORY = 'invenio_files_rest.permissions.permission_factory'**

Permission factory to control the files access from the REST interface.

invenio_files_rest.config.**FILES_REST_SIZE_LIMITERS = 'invenio_files_rest.limiters.file_size_limiters'**

Import path of file size limiters factory to control bucket size limits.

invenio_files_rest.config.**FILES_REST_STORAGE_CLASS_LIST = {'A': 'Archive', 'S': 'Standard'}**

Storage class list defines the systems storage classes.

Storage classes are useful for e.g. defining the type of storage an object is located on (e.g. offline/online), so that the system known if it can serve the file and/or what is the reliability.

invenio_files_rest.config.**FILES_REST_STORAGE_FACTORY = 'invenio_files_rest.storage.pyfs_storage_factory'**

Import path of factory used to create a storage instance.

invenio_files_rest.config.**FILES_REST_STORAGE_PATH_DIMENSIONS = 2**

Number of directory levels created when generating the path of a file.

For example, if split length set to 2 and dimension to 3, the final path will be *a2/ad/4k/c9-8j39-34jn/*.

invenio_files_rest.config.**FILES_REST_STORAGE_PATH_SPLIT_LENGTH = 2**

Number of chars to use as folder name when generating the path of a file.

For example, if split length set to 4 and dimension to 4, the final path will be *a2ad/4kc9/8j39-34jn/*.

invenio_files_rest.config.**FILES_REST_TASK_WAIT_INTERVAL = 2**

Interval in seconds between sending a whitespace to not close connection.

invenio_files_rest.config.**FILES_REST_TASK_WAIT_MAX_SECONDS = 600**

Maximum number of seconds to wait for a task to finish.

invenio_files_rest.config.**FILES_REST_UPLOAD_FACTORIES = ['invenio_files_rest.views:stream_uploadfactory', 'invenio_files_rest.views:ngfileupload_uploadfactory']**

Import path of factories used when parsing upload parameters.

Note: Factories that reads `request.stream` directly must be first in the list, otherwise Werkzeug's form-data parser will read the stream.

---

invenio_files_rest.config.**FILES_REST_XSENDFILE_ENABLED = False**

Use the X-Accel-Redirect header to stream the file through a reverse proxy(e.g NGINX).

invenio_files_rest.config.**FILES_REST_XSENDFILE_RESPONSE_FUNC**(*obj*)

Function for the creation of a file streaming redirect response.

invenio_files_rest.config.**MAX_CONTENT_LENGTH = 16777216**

Maximum allowed content length for form data.

This value limits the maximum file upload size via multipart-formdata and is a Flask configuration variable that by default is unlimited. The value must be larger than the maximum part size you want to accept via application/multipart-formdata (used by e.g. ng-file upload). This value only limits file upload size via application/multipart-formdata and in particular does not restrict the maximum file size possible when streaming a file in the body of a PUT request.

Flask, by default, saves any file bigger than 500kb to a temporary file on disk, thus do not set this value to large or you may run out of disk space on your nodes.

1.4 Usage

Invenio-Files-REST module.

This guide will show you how to get started with Invenio-Files-REST. It assumes that you already have knowledge of Flask applications and Invenio modules.

It will then explain key topics and concepts of this module.

1.4.1 Getting started

You will learn how to create a new Location, a Bucket and an ObjectVersion using the programmatic APIs of Invenio-Files-REST.

First, you will have to setup your virtualenv environment and install this module along with all it's dependencies.

After that, start a Python shell and execute the following commands:

```
>>> from flask import Flask
>>> app = Flask('myapp')
```

This is the initial configuration needed to have things running:

```
>>> app.config['BROKER_URL'] = 'redis://'
>>> app.config['CELERY_RESULT_BACKEND'] = 'redis://'
>>> app.config['DATADIR'] = 'data'
>>> app.config['FILES_REST_MULTIPART_CHUNKSIZE_MIN'] = 4
>>> app.config['REST_ENABLE_CORS'] = True
>>> app.config['SECRET_KEY'] = 'CHANGEME'
>>> app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite://'
>>> app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
>>> allow_all = lambda *args, **kwargs: \
... type('Allow', (), {'can': lambda self: True})()
>>> app.config['FILES_REST_PERMISSION_FACTORY'] = allow_all
```

Relevant configuration variables will be explained later on. Now let's initialize all required Invenio extensions:

```
>>> import shutil
>>> from os import makedirs
>>> from os.path import dirname, exists, join
>>> from pprint import pprint
>>> import json
```

```
>>> from invenio_i18n import Babel
>>> from flask_menu import Menu
>>> from invenio_db import InvenioDB, db
>>> from invenio_rest import InvenioREST
>>> from invenio_admin import InvenioAdmin
>>> from invenio_accounts import InvenioAccounts
>>> from invenio_access import InvenioAccess
>>> from invenio_accounts.views import blueprint as accounts_blueprint
>>> from invenio_celery import InvenioCelery
>>> from invenio_files_rest import InvenioFilesREST
>>> from invenio_files_rest.views import blueprint
```

```
>>> ext_babel = Babel(app)
>>> ext_menu = Menu(app)
>>> ext_db = InvenioDB(app)
>>> ext_rest = InvenioREST(app)
>>> ext_admin = InvenioAdmin(app)
>>> ext_accounts = InvenioAccounts(app)
>>> ext_access = InvenioAccess(app)
```

You can now initialize Invenio-Files-REST. When using Invenio-Files-REST as dependency of an Invenio applicaton, the REST views are automatically registered via entry points. For this example, you will have to register them manually and push a Flask application context:

```
>>> ext_rest = InvenioFilesREST(app)
```

```
>>> app.register_blueprint(accounts_blueprint)
>>> app.register_blueprint(blueprint)
```

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

Let's create the database and tables, using an in-memory SQLite database:

```
>>> db.create_all()
```

When you setup Invenio-Files-REST for the first time, you will have to define a default Location. It can be local or remote and it will be accessed via its URI.

Create a location

For this example, you will use a temporary directory:

```
>>> from invenio_files_rest.models import Location
>>> d = app.config['DATADIR']  # folder `data`
>>> if exists(d): shutil.rmtree(d)
>>> makedirs(d)
>>> loc = Location(name='local', uri=d, default=True)
>>> db.session.add(loc)
>>> db.session.commit()
```

Create a bucket

In order to create, modify or delete files, you have to create a files container first, the Bucket.

```
>>> from invenio_files_rest.models import Bucket
>>> b1 = Bucket.create(loc)
>>> db.session.commit()
```

Create objects

Files are represented by ObjectVersions. After creating a bucket, you can now add files to it, for example:

```
>>> from io import BytesIO
>>> from invenio_files_rest.models import ObjectVersion
>>> a_file = BytesIO(b"my file contents")
>>> f = ObjectVersion.create(b1, "thesis.pdf", stream=a_file)
>>> db.session.commit()
```

Retrieve objects

You can now retrieve objects. Retrieve the bucket object:

```
>>> b = Bucket.get(b1.id)
```

Retrieve all ObjectVersions contained in a bucket:

```
>>> file_names = [ov.key for ov in ObjectVersion.get_by_bucket(b1.id)]
```

Retrieve a specific ObjectVersion by filename:

```
>>> f = ObjectVersion.get(b1.id, "thesis.pdf")
```

1.4.2 Data model

This is a more in-depth explanation of the concepts introduced in the *Overview* section.

Buckets

A bucket is a container of objects. It is uniquely identified by an ID. Buckets have a default Location and Storage class. Individual objects in the bucket can however have different Locations and Storage classes.

The `size` field stores the current size of the bucket. When a new object is added or completely removed, its size is updated.

Buckets can have constraints on the maximum amount of objects that they can contain. It is controlled by the function *invenio_files_rest.limiters.file_size_limiters()*: by default, a new object can be added to the bucket if the maximum size of the file is lower than *invenio_files_rest.config.FILES_REST_DEFAULT_MAX_FILE_SIZE* and if the total quota (the sum of sizes of all files) is lower than *invenio_files_rest.config.FILES_REST_DEFAULT_QUOTA_SIZE*.

Buckets can be marked as `locked`. When a bucket is locked, objects can be retrieved but no object can be added and deleted.

Similarly to objects, bucket can be logically marked as `deleted` without affecting the actual content. When it is deleted, it simply means that no objects can be retrieved or added via APIs.

Finally, buckets provide ways to create or synchronize copies: the `snapshot` operation creates a new copy of a bucket with all the latest versions of the object it contains, without duplicating files on disk. The `sync` operation mirrors objects contained in the source bucket to the destination bucket.

ObjectVersion

ObjectVersions are objects that represent a specific version of a file at a given point in time. ObjectVersions are uniquely identified by its ID. They are always contained in an existing Bucket by having the reference `bucket_id` to it.

An ObjectVersion describes the file (`FileInstance`) that references with the attribute `file_id`. It also stores some metadata of the file: the file name, stored in the `key` attribute and the version, stored in `version_id` attribute. The triplet (bucket_id, key, version_id) is unique.

For a given `key` in a Bucket, normally the latest version in history is marked as the `head`.

The `key` has a maximum length defined via *invenio_files_rest.config.FILES_REST_OBJECT_KEY_MAX_LEN*.

ObjectVersion can be marked as deleted by removing its reference to the file it represents: from the user perspective, deleting a file normally means adding a new ObjectVersion, which will be the new `head`, without `file_id`.

FileInstance

A file instance represents a file on disk. A file instance may be linked from many objects, while an object can have one and only one file instance.

The file on disk can be retrieved by the file instance `uri`, which is an absolute path/URI generated when adding the file: the base path is retrieved from the `Location` used for this file, and the relative path is assigned by the file's `Storage`. It is responsibility of the Storage, which is aware of the file system that is managing, to generate a unique final path for the file. You can modify how the path is generated with the default storage *invenio_files_rest.storage.pyfs_storage_factory()* by changing *invenio_files_rest.config.FILES_REST_STORAGE_PATH_SPLIT_LENGTH* or *invenio_files_rest.config.FILES_REST_STORAGE_PATH_DIMENSIONS*.

A file instance may not be ready to be accessed, for example in case of multipart uploads: the attribute `readable` marks it. It can also be marked as not `writable` if it cannot be deleted or replaced, for safety reasons.

`checksum`, `last_check_at` and `last_check` are attributes used to store information about integrity checks.

You can find the documentation of each API in the *API Docs*.

1.4.3 REST APIs

REST APIs allow you to perform most of the operations needed when manipulating files.

By design, Locations cannot be created using REST APIs. This is because they depend on your physical file storage infrastructure. You will have to create them in advance when setting up your Invenio instance.

To be able to run each of the next steps, you can instantiate and start an Invenio instance as described here.

Create a bucket

A bucket can be created by a POST request to `/files`. The response will contain the unique ID of the bucket.

```
$ curl -X POST http://localhost:5000/api/files
```

```
{
    "max_file_size": null,
    "updated": "2019-05-16T13:07:21.595398+00:00",
    "locked": false,
    "links": {
        "self": "http://localhost:5000/api/files/
                 cb8d0fa7-2349-484b-89cb-16573d57f09e",
        "uploads": "http://localhost:5000/api/files/
                    cb8d0fa7-2349-484b-89cb-16573d57f09e?uploads",
        "versions": "http://localhost:5000/api/files/
                     cb8d0fa7-2349-484b-89cb-16573d57f09e?versions"
    },
    "created": "2019-05-16T13:07:21.595391+00:00",
    "quota_size": null,
    "id": "cb8d0fa7-2349-484b-89cb-16573d57f09e",
    "size": 0
}
```

Uploading Files

You can upload, download and modify single files via REST APIs. A file is uniquely identified within a bucket by its name and version. Each file can have multiple versions.

Let's upload a file called `my_file.txt` inside the bucket that was just created.

```
$ BUCKET=cb8d0fa7-2349-484b-89cb-16573d57f09e

$ echo "my file content" > my_file.txt

$ curl -i -X PUT --data-binary @my_file.txt \
  "http://localhost:5000/api/files/$BUCKET/my_file.txt"
```

```
{
    "mimetype": "text/plain",
    "updated": "2019-05-16T13:10:22.621533+00:00",
    "links": {
        "self": "http://localhost:5000/api/files/
                 cb8d0fa7-2349-484b-89cb-16573d57f09e/my_file.txt",

        "version": "http://localhost:5000/api/files/
                    cb8d0fa7-2349-484b-89cb-16573d57f09e/my_file.txt?
                    versionId=7f62676d-0b8e-4d77-9687-8465dc506ca8",
        "uploads": "http://localhost:5000/api/files/
                    cb8d0fa7-2349-484b-89cb-16573d57f09e/
                    my_file.txt?uploads"
    },
    "is_head": true,
    "tags": {},
    "checksum": "md5:d7d02c7125bdcdd857eb70cb5f19aecc",
    "created": "2019-05-16T13:10:22.617714+00:00",
    "version_id": "7f62676d-0b8e-4d77-9687-8465dc506ca8",
    "delete_marker": false,
    "key": "my_file.txt",
    "size": 14
}
```

If you have a new version of the file, you can upload it to the same bucket using the same filename. In this case, a new ObjectVersion will be created.

```
$ echo "my file content version 2" > my_filev2.txt

$ curl -i -X PUT --data-binary @my_filev2.txt \
  "http://localhost:5000/api/files/$BUCKET/my_file.txt"
```

```
{
    "mimetype": "text/plain",
    "updated": "2019-05-16T13:11:22.621533+00:00",
    "links": {
        "self": "http://localhost:5000/api/files/
                 cb8d0fa7-2349-484b-89cb-16573d57f09e/my_file.txt",
```

(continues on next page)

(continued from previous page)

```
        "version": "http://localhost:5000/api/files/
                    cb8d0fa7-2349-484b-89cb-16573d57f09e/my_file.txt?
                    versionId=24bf075f-09f4-42f8-9fbe-3f00b8aac3e8",
        "uploads": "http://localhost:5000/api/files/
                    cb8d0fa7-2349-484b-89cb-16573d57f09e/
                    my_file.txt?uploads"
    },
    "is_head": true,
    "tags": {},
    "checksum": "md5:fe76512703258a894e56bac89d2e8dec",
    "created": "2019-05-16T13:11:22.617714+00:00",
    "version_id": "24bf075f-09f4-42f8-9fbe-3f00b8aac3e8",
    "delete_marker": false,
    "key": "my_file.txt",
    "size": 13
}
```

When integrating the REST APIs to upload files via a web application, you might use JavaScript to improve user experience. Invenio-Files-REST provides out of the box integration with JavaScript uploaders. See the *JS Uploaders* section for more information.

Invenio-Files-REST also provides different ways to upload large files. See the *Multipart Upload* and *Large Files* sections for more information.

Serving files

To serve and allow download of files, you can perform a GET request specifying the bucket and the filename used to upload the file.

```
$ curl -i -X GET "http://localhost:5000/api/files/$BUCKET/my_file.txt"
```

You can also list files or download specific versions of files. See the REST APIs reference documentation below for more information.

Be aware that there are security implications to take into account when serving files. See the *Security* for more information.

Invenio-Files-Rest provides also the functionality to serve your files directly from your external storage. This is achieved by attaching the X-Accel-Redirect header to the response, which will then be redirected by your Web Proxy (e.g. NGINX, Apache) to your external storage, finally streaming the file directly to the user. To use this feature you will need to configure your Web Proxy accordingly and then enable the *invenio_files_rest.config.FILES_REST_XSENDFILE_ENABLED*.

API Reference

Default Location

Create a bucket:

```
POST /files/
```

Buckets

Check if bucket exists, returning either a 200 or 404:

```
HEAD /files/<bucket_id>
```

Retrieve the latest version of all objects in bucket:

```
GET /files/<bucket_id>
```

Retrieve all versions of files in a bucket:

```
GET /files/<bucket_id>?versions
```

Return list of multipart uploads:

```
GET /files/<bucket_id>?uploads
```

ObjectVersions

Initiate multipart upload (see *Multipart Upload*):

```
POST /files/<bucket_id>/<file_name>?
     uploads&size=<total_size>&partSize=<part_size>
```

Finalize multipart upload:

```
POST /files/<bucket_id>/<file_name>?uploadId=<upload_id>
```

Upload a file to a bucket:

```
PUT /files/<bucket_id>/<file_name>
```

Upload part of in-progress multipart upload to a bucket:

```
PUT /files/<bucket_id>/<file_name>?uploadId=<upload_id>&part=<part_number>
```

Retrieve the latest version of a given file. By default, the file is returned with the header `'Content-Disposition': 'inline'`. Be aware that the browser will try to preview it.

```
GET /files/<bucket_id>/<file_name>
```

Download the latest version of a given file. It will return the same response as the request above but with the response header `'Content-Disposition': 'attachment'` to instruct the browser trigger a download.

```
GET /files/<bucket_id>/<file_name>?download
```

Retrieve a specific version of a given file:

```
GET /files/<bucket_id>/<file_name>?versionId=<version_id>
```

Retrieve the list of parts of a multipart upload:

```
GET /files/<bucket_id>/<file_name>?uploadId=<id_number>
```

Mark an object as deleted (see *Deleting files*):

```
DELETE /files/<bucket_id>/<file_name>
```

Permanently erase an object and the physical file on disk:

```
DELETE /files/<bucket_id>/<file_name>?versionId=<version_id>
```

Abort multipart upload:

```
DELETE /files/<bucket_id>/<file_name>?uploadId=<upload_id>
```

1.4.4 Deleting files

A delete operation can be of two types:

1. mark an object as deleted, allowing the possibility of restoring a deleted file (also called delete marker or soft deletion).
2. permanently remove any trace of an object and referenced file on disk (also called hard deletion).

Soft deletion

Technically, it creates a new ObjectVersion, that becomes the new `head`, with no reference to a FileInstance. It is possible to revert it by getting the previous version.

This operation will not access to the file on disk and it will leave it untouched.

You can soft delete using REST APIs:

```
DELETE /files/<bucket_id>/<file_name>
```

Hard deletion

Given a specific object version, it will delete the ObjectVersion, the referenced FileInstance and the file on disk. If the deleted version was the `head`, it will then set the previous object as the new head.

The deletion of files on disk will not happen immediately. This is because it is done via an asynchronous task to ensure that the FileInstance is safely removed from the database in case the low level operation of file removal on disk fails for any unexpected reason.

You can hard delete a file using REST APIs:

```
DELETE /files/<bucket_id>/<file_name>?versionId=<version_id>
```

REST APIs do not allow to perform delete operations that can affect multiple objects at the same time. For advanced use cases, you will to use the Invenio-Files-REST APIs programmatically.

Note: For safety reasons, the deletion will fail if the file that you want to delete is referenced by multiple ObjectVersions, for example in case of Buckets snapshots.

1.4.5 Authorization

Invenio-Files-REST relies on Invenio-Access to implement files authorization. The following documentation assumes that you already have knowledge of how authorization works on Invenio.

Invenio-Files-REST defines a set of actions for operations on Bucket and ObjectVersions that can be used to implement authorization as you need:

- files-rest-location-update
- files-rest-bucket-read
- files-rest-bucket-read-versions
- files-rest-bucket-update
- files-rest-bucket-listmultiparts
- files-rest-object-read
- files-rest-object-read-version
- files-rest-object-delete
- files-rest-object-delete-version
- files-rest-multipart-read
- files-rest-multipart-delete

Response codes

If the authorization for an action fails, Invenio-Files-REST normally returns a `403` response code for authenticated users, `401` otherwise. For security reasons, when trying to retrieve an unauthorized file, it will return a `404` instead to hide the existence or non-existence of the file.

Authorization definition

The default permission factory *invenio_files_rest.permissions.permission_factory* will authorize users that has `Needs` that fulfill the actions listed above. This means that by default no user will be authorized (with the exception of any `superuser`).

Depending on how you are planning to integrate Invenio-Files-REST in your Invenio application, you might want to decide how to give permissions for operations on files.

If you plan to give authorization to specific users or roles, you can use the default permission factory and assign user or roles to the actions listed above as described in the Invenio-Access documentation.

If instead you want to define permissions based on other object, for example on records to which the files are attached to, then you will have to define your own permission factory and used via the configuration variable *invenio_files_rest.config.FILES_REST_PERMISSION_FACTORY*.

See *invenio_files_rest.permissions* for more documentation.

1.4.6 Security

When serving files, you will have to take into account any security implications. Here you can find some recommendations to mitigate possible vulnerabilities, such as Cross-Site Scripting (XSS):

1. If possible, serve user uploaded files from a separate domain (not a subdomain).
2. By default, Invenio-Files-REST sets some response headers to prevent the browser from rendering and executing HTML files. See *invenio_files_rest.helpers.send_stream()* for more information.
3. Prefer file download instead of allowing the browser to preview any file, by adding the `?download` URL query argument

1.4.7 Signals

Invenio-Files-REST supports signals that can be used to react to events.

Events are sent whenever a file is downloaded, uploaded or deleted.

As an example, let's listen to the file download event:

```
from invenio_files_rest.signals import file_downloaded

def after_file_downloaded(event, sender_app, obj=None, **kwargs):
    print("File downloaded {0}".format(obj))

listener = file_downloaded.connect(after_file_downloaded)
# Request to download a file for the event to trigger
```

See *invenio_files_rest.signals* for more documentation.

Integrity

Invenio-Files-REST computes and stores checksums when files are uploaded and it allows you to set up periodic tasks to regularly re-validate files integrity.

By default, it uses `MD5` to compute checksums. You can override this by subclassing *invenio_files_rest.storage.FileStorage*.

You can use the tasks *invenio_files_rest.tasks.verify_checksum()* and *invenio_files_rest.tasks.schedule_checksum_verification()* to set up periodic tasks to perform checksum verifications on single files or batches and provide reports.

Let's create a periodic task to compute checksums:

```
CELERY_BEAT_SCHEDULE = {
    'file-checks': {
       'task': 'invenio_files_rest.tasks.schedule_checksum_verification',
       'schedule': timedelta(hours=1),
    }
}
```

By default, *invenio_files_rest.tasks.schedule_checksum_verification()* will generate batches of files to check using some predefined constraints, in order to throttle the execution rate of the checks. It will then spawn a celery task *invenio_files_rest.tasks.verify_checksum()* for each of the file in the set.

You can customize most of these parameters by passing the method arguments to the schedule definition.

Keep in mind that you need to have `celerybeat` running.

1.4.8 Storage Backends

Invenio-Files-REST provides a default implementation of storage factory *invenio_files_rest.storage.PyFSFileStorage* used when performing operation on files in the defined locations. The `PyFSFileStorage` class uses PyFilesystem to access the file system.

Build your own Storage Backend

In order to use a different storage backend, you can implement the *invenio_files_rest.storage.FileStorage* interface.

Mandatory methods to implement:

- `initialize`
- `open`
- `save`
- `update`
- `delete`

Optional methods to implement:

- `send_file`
- `checksum`
- `copy`
- `_init_hash`
- `_compute_checksum`
- `_write_stream`

Then, you will have to re-implement a storage factory in a similar way as the default *invenio_files_rest.storage.pyfs_storage_factory()* and set configuration variable *invenio_files_rest.config.FILES_REST_STORAGE_FACTORY*.

1.4.9 JS Uploaders

Some JS uploaders do not allow you to customize the HTTP request that is sent to the REST APIs when uploading a file. If the default implementation provided by Invenio-Files-REST is not compatible, you will have to implement your own custom factory to adapt the JS uploader request to Invenio-Files-REST.

When using the AngularJS uploader ng-file-upload, Invenio-Files-REST already provides a compatible factory, *invenio_files_rest.views.ngfileupload_uploadfactory()*.

If you have to create a new custom factory, you have to:

1. Create your own factory similar to *invenio_files_rest.views.ngfileupload_uploadfactory()*.

2. Instruct Invenio-Files-REST to use it by setting the configuration variables *invenio_files_rest.config.FILES_REST_MULTIPART_PART_FACTORIES* and *invenio_files_rest.config.FILES_REST_UPLOAD_FACTORIES*

1.4.10 Multipart Upload

You might want to optimize upload in case of large files. Invenio-Files-REST allows you to upload parts of the same file in parallel via multiparts uploads.

A multipart upload requires that each part of the file has the same size, except for the last one that can be smaller. Each part can be uploaded at the same time and at the end of the process all parts are merged into one single file.

In case of failure when uploading one of the parts, the operation is completely aborted and all parts are deleted.

With Invenio-Files-REST, the multipart upload consists of 3 actions:

- An initial request to initiate the upload and obtain an `id` to be used for each part upload.
- A series of requests to upload of each part specifying the part number to correctly merge the file at the end.
- A final request to to merge all parts together.

Let's see an example. Let's create an 11 MB file which will then be split into 2 chunks using the linux `split` command:

```
$ dd if=/dev/urandom of=my_file.txt bs=1048576 count=11

$ split -b6291456 my_file.txt segment_
```

Create a new bucket:

```
$ curl -X POST http://localhost:5000/api/files
```

Response:

```
{
   "max_file_size":null,
   "updated":"2019-05-17T06:52:52.897378+00:00",
   "locked":false,
   "links":{
      "self":"http://localhost:5000/api/files/
              c896d17b-0e7d-44b3-beba-7e43b0b1a7a4",
      "uploads":"http://localhost:5000/api/files/
                 c896d17b-0e7d-44b3-beba-7e43b0b1a7a4?uploads",
      "versions":"http://localhost:5000/api/files/
                  c896d17b-0e7d-44b3-beba-7e43b0b1a7a4?versions"
   },
   "created":"2019-05-17T06:52:52.897373+00:00",
   "quota_size":null,
   "id":"c896d17b-0e7d-44b3-beba-7e43b0b1a7a4",
   "size":0
}
```

Now, let's initiate the multipart upload. Notice the URL query argument that specify total size and each part size:

```
$ B=c896d17b-0e7d-44b3-beba-7e43b0b1a7a4

$ curl -i -X POST \
  "http://localhost:5000/api/files/$B/my_file.txt?
   uploads&size=11534336&partSize=6291456"
```

Notice the upload `id` in the response:

```
{
   "updated":"2019-05-17T07:07:22.219002+00:00",
   "links":{
      "self":"http://localhost:5000/api/files/
              c896d17b-0e7d-44b3-beba-7e43b0b1a7a4/my_file.txt?
              uploadId=a85b1cbd-4080-4c81-a95c-b4df5d1b615f",

      "object":"http://localhost:5000/api/files/
                c896d17b-0e7d-44b3-beba-7e43b0b1a7a4/my_file.txt",

      "bucket":"http://localhost:5000/api/files/
                c896d17b-0e7d-44b3-beba-7e43b0b1a7a4"
   },
   "last_part_size":5242880,
   "created":"2019-05-17T07:07:22.218998+00:00",
   "bucket":"c896d17b-0e7d-44b3-beba-7e43b0b1a7a4",
   "completed":false,
   "part_size":6291456,
   "key":"my_file.txt",
   "last_part_number":1,
   "id":"a85b1cbd-4080-4c81-a95c-b4df5d1b615f",
   "size":11534336
}
```


Now, let's upload each part in parallel. Notice the `uploadId` and `partNumber` URL query arguments:

```
$ U=a85b1cbd-4080-4c81-a95c-b4df5d1b615f

$ curl -i -X PUT --data-binary @segment_aa \
  "http://localhost:5000/api/files/$B/my_file.txt?uploadId=$U&partNumber=0"

$ curl -i -X PUT --data-binary @segment_ab \
  "http://localhost:5000/api/files/$B/my_file.txt?uploadId=$U&partNumber=1"
```

Complete the multipart upload:

```
$ curl -i -X POST \
  "http://localhost:5000/api/files/$B/my_file.txt?uploadId=$U"
```

You can also abort a multipart upload (and delete all uploaded parts):

```
$ curl -i -X DELETE \
  "http://localhost:5000/api/files/$B/my_file.txt?uploadId=$U"
```

Multiparts uploads limits can be controlled via configuration variables:

- Set *invenio_files_rest.config.FILES_REST_MULTIPART_MAX_PARTS* to limit the maximum number of parts for a single multipart upload.
- Set *invenio_files_rest.config.FILES_REST_MULTIPART_CHUNKSIZE_MIN* to define the minimum size of each part.
- Set *invenio_files_rest.config.FILES_REST_MULTIPART_CHUNKSIZE_MAX* to define the maximum size of each part.

- Set *invenio_files_rest.config.FILES_REST_MULTIPART_EXPIRES* to define the maximum number of days for which a multipart upload is considered valid and accepts new part uploads.

1.4.11 Large Files

By default, Flask and your web server have a limit on the maximum size of the upload files. Normally, when the max size is exceeded, the server will return a response code `413 (Request Entity Too Large)`.

You can adjust these configurations according to your needs.

For Flask, specify `MAX_CONTENT_LENGTH` configuration variable. Be aware that if the request does not specify a `CONTENT_LENGTH`, no data will be read. To change the max size, you can for example:

```
$ app.config['MAX_CONTENT_LENGTH'] = 25 * 1024 * 1024
```

Here is an example for Nginx web server. If you are using another web server, please check the related documentation.

```
http {
    ...
    client_max_body_size 25M;
}
```

1.4.12 Data Migration

When you already have an instance running with a certain amount of uploaded data, you might have the need to migrate the data to a different, larger or more efficient physical location. It can involve your entire set of files or just a part of it.

Note that files migration can be performed with no downtime and in a completely transparent way for the user.

The steps to perform a complete migration are the followings:

1. Create the new `Location` in the database with the URI of your new location and set it to `default = True`. In this way, new `Buckets` will use the new default location.
2. Change all existing buckets locations in the database to the new one. By doing this, any new file uploaded to the existing bucket will be stored in the new location.
3. For each `FileInstance`, run the asynchronous task *invenio_files_rest.tasks.migrate_file()* passing the new location.

The asynchronous task *invenio_files_rest.tasks.migrate_file()* will create a new `FileInstance` and copy the file content to the new location. It will then change each `ObjectVersion` that have a reference to the old `FileInstance` to reference the new `FileInstance` and eventually run an integrity check.

CHAPTER

TWO

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

Files download/upload REST API similar to S3 for Invenio.

class invenio_files_rest.ext.**InvenioFilesREST**(*app=None*)

Invenio-Files-REST extension.

Extension initialization.

init_app(*app*)

Flask application initialization.

init_config(*app*)

Initialize configuration.

2.1.1 Models

Models for Invenio-Files-REST.

The entities of this module consists of:

- **Buckets** - Identified by UUIDs, and contains objects.
- **Buckets tags** - Identified uniquely with a bucket by a key. Used to store extra metadata for a bucket.
- **Objects** - Identified uniquely within a bucket by string keys. Each object can have multiple object versions (note: Objects do not have their own database table).
- **Object versions** - Identified by UUIDs and belongs to one specific object in one bucket. Each object version has zero or one file instance. If the object version has no file instance, it is considered a *delete marker*.
- **File instance** - Identified by UUIDs. Represents a physical file on disk. The location of the file is specified via a URI. A file instance can have many object versions.
- **Locations** - A bucket belongs to a specific location. Locations can be used to represent e.g. different storage systems.
- **Multipart Objects** - Identified by UUIDs and belongs to a specific bucket and key.
- **Part object** - Identified by their multipart object and a part number.

The actual file access is handled by a storage interface. Also, objects do not have their own model, but are represented via the *ObjectVersion* model.

class invenio_files_rest.models.**Bucket**(***kwargs*)

Model for storing buckets.

A bucket is a container of objects. Buckets have a default location and storage class. Individual objects in the bucket can however have different locations and storage classes.

A bucket can be marked as deleted. A bucket can also be marked as locked to prevent operations on the bucket.

Each bucket can also define a quota. The size of a bucket is the size of all objects in the bucket (including all versions).

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.

classmethod all()

Return query of all buckets (excluding deleted).

classmethod create(*location=None*, *storage_class=None*, ***kwargs*)

Create a bucket.

Parameters

- **location** – Location of a bucket (instance or name). Default: Default location.
- **storage_class** – Storage class of a bucket. Default: Default storage class.
- ****kwargs** – Keyword arguments are forwarded to the class
- ****kwargs** – Keyword arguments are forwarded to the class constructor.

Returns
Created bucket.

created

Creation timestamp.

default_location

Default location.

default_storage_class

Default storage class.

classmethod delete(*bucket_id*)

Delete a bucket.

Does not actually delete the Bucket, just marks it as deleted.

deleted

Delete state of bucket.

classmethod get(*bucket_id*)

Get a bucket object (excluding deleted).

Parameters
bucket_id – Bucket identifier.

Returns
Bucket instance.

get_tags()

Get tags for bucket as dictionary.

id

Bucket identifier.

location

Location associated with this bucket.

locked

Lock state of bucket.

Modifications are not allowed on a locked bucket.

max_file_size

Maximum size of a single file in the bucket.

Usage of this property depends on which file size limiters are installed.

property quota_left

Get how much space is left in the bucket.

quota_size

Quota size of bucket.

Usage of this property depends on which file size limiters are installed.

remove()

Permanently remove a bucket and all objects (including versions).

> **Warning:** This by-passes the normal versioning and should only be used when you want to permanently delete a bucket and its objects. Otherwise use *Bucket.delete()*.
>
> Note the method does not remove the associated file instances which must be garbage collected.

Returns
self.

size

Size of bucket.

This is a computed property which can rebuilt any time from the objects inside the bucket.

property size_limit

Get size limit for this bucket.

The limit is based on the minimum output of the file size limiters.

snapshot(*lock=False*)

Create a snapshot of latest objects in bucket.

Parameters
lock – Create the new bucket in a locked state.

Returns
Newly created bucket containing copied ObjectVersion.

sync(*bucket*, *delete_extras=False*)

Sync self bucket ObjectVersions to the destination bucket.

The bucket is fully mirrored with the destination bucket following the logic:

- same ObjectVersions are not touched
- new ObjectVersions are added to destination
- deleted ObjectVersions are deleted in destination
- extra ObjectVersions in dest are deleted if *delete_extras* param is True

Parameters

- **bucket** – The destination bucket.
- **delete_extras** – Delete extra ObjectVersions in destination if True.

Returns

The bucket with an exact copy of ObjectVersions in self.

updated

Modification timestamp.

validate_storage_class(*key*, *default_storage_class*)

Validate storage class.

class invenio_files_rest.models.**BucketTag**(***kwargs*)

Model for storing tags associated to buckets.

This is useful to store extra information for a bucket.

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 buckets.

classmethod create(*bucket*, *key*, *value*)

Create a new tag for bucket.

classmethod create_or_update(*bucket*, *key*, *value*)

Create or update a new tag for bucket.

classmethod delete(*bucket*, *key*)

Delete a tag.

classmethod get(*bucket*, *key*)

Get tag object.

classmethod get_value(*bucket*, *key*)

Get tag value.

key

Tag key.

value

Tag value.

class invenio_files_rest.models.**FileInstance**(***kwargs*)

Model for storing files.

A file instance represents a file on disk. A file instance may be linked from many objects, while an object can have one and only one file instance.

A file instance also records the storage class, size and checksum of the file on disk.

Additionally, a file instance can be read only in case the storage layer is not capable of writing to the file (e.g. can typically be used to link to files on externally controlled storage).

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.

checksum

String representing the checksum of the object.

clear_last_check()

Clear the checksum of the file.

copy_contents(*fileinstance*, *progress_callback=None*, *chunk_size=None*, ***kwargs*)

Copy this file instance into another file instance.

classmethod create()

Create a file instance.

Note, object is only added to the database session.

created

Creation timestamp.

delete()

Delete a file instance.

The file instance can be deleted if it has no references from other objects. The caller is responsible to test if the file instance is writable and that the disk file can actually be removed.

Note: Normally you should use the Celery task to delete a file instance, as this method will not remove the file on disk.

---

classmethod get(*file_id*)

Get a file instance.

classmethod get_by_uri(*uri*)

Get a file instance by URI.

id

Identifier of file.

init_contents(*size=0*, ***kwargs*)

Initialize file.

last_check

Result of last fixity check.

last_check_at

Timestamp of last fixity check.

readable

Defines if the file is read only.

send_file(*filename*, *restricted=True*, *mimetype=None*, *trusted=False*, *chunk_size=None*, *as_attachment=False*, ***kwargs*)

Send file to client.

set_contents(*stream*, *chunk_size=None*, *size=None*, *size_limit=None*, *progress_callback=None*, ***kwargs*)

Save contents of stream to this file.

Parameters

- **obj** – ObjectVersion instance from where this file is accessed from.
- **stream** – File-like stream.

set_uri(*uri*, *size*, *checksum*, *readable=True*, *writable=False*, *storage_class=None*)

Set a location of a file.

size

Size of file.

storage(***kwargs*)

Get storage interface for object.

Uses the applications storage factory to create a storage interface that can be used for this particular file instance.

Returns

Storage interface.

storage_class

Storage class of file.

update_checksum(*progress_callback=None*, *chunk_size=None*, *checksum_kwargs=None*, ***kwargs*)

Update checksum based on file.

update_contents(*stream*, *seek=0*, *size=None*, *chunk_size=None*, *progress_callback=None*, ***kwargs*)

Save contents of stream to this file.

Parameters

- **obj** – ObjectVersion instance from where this file is accessed from.
- **stream** – File-like stream.

updated

Modification timestamp.

uri

Location of file.

validate_uri(*key*, *uri*)

Validate uri.

verify_checksum(*progress_callback=None*, *chunk_size=None*, *throws=True*, *checksum_kwargs=None*, ***kwargs*)

Verify checksum of file instance.

Parameters

- **throws** (*bool*) – If *True*, exceptions raised during checksum calculation will be re-raised after logging. If set to *False*, and an exception occurs, the *last_check* field is set to *None* (*last_check_at* of course is updated), since no check actually was performed.
- **checksum_kwargs** (*dict*) – Passed as ***kwargs* ` to `storage().checksum`.

writable

Defines if file is writable.

This property is used to create a file instance prior to having the actual file at the given URI. This is useful when e.g. copying a file instance.

class invenio_files_rest.models.**Location**(***kwargs*)

Model defining base locations.

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.

classmethod all()

Return query that fetches all locations.

created

Creation timestamp.

default

True if the location is the default location.

At least one location should be the default location.

classmethod get_by_name(*name*)

Fetch a specific location object.

classmethod get_default()

Fetch the default location object.

id

Internal identifier for locations.

The internal identifier is used only used as foreign key for buckets in order to decrease storage requirements per row for buckets.

name

External identifier of the location.

updated

Modification timestamp.

uri

URI of the location.

validate_name(*key*, *name*)

Validate name.

class invenio_files_rest.models.**MultipartObject**(***kwargs*)

Model for storing files in chunks.

A multipart object belongs to a specific bucket and key and is identified by an upload id. You can have multiple multipart uploads for the same bucket and key. Once all parts of a multipart object is uploaded, the state is changed to `completed`. Afterwards it is not possible to upload new parts. Once completed, the multipart object is merged, and added as a new version in the current object/bucket.

All parts for a multipart upload must be of the same size, except for the last part.

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 buckets.

bucket_id

Bucket identifier.

chunk_size

Size of chunks for file.

complete()

Mark a multipart object as complete.

completed

Defines if object is the completed.

classmethod create(*bucket*, *key*, *size*, *chunk_size*)

Create a new object in a bucket.

created

Creation timestamp.

delete()

Delete a multipart object.

expected_part_size(*part_number*)

Get expected part size for a particular part number.

file

Relationship to buckets.

file_id

File instance for this multipart object.

classmethod get(*bucket*, *key*, *upload_id*, *with_completed=False*)

Fetch a specific multipart object.

static is_valid_chunksize(*chunk_size*)

Check if size is valid.

static is_valid_size(*size*, *chunk_size*)

Validate max theoretical size.

key

Key identifying the object.

property last_part_number

Get last part number.

property last_part_size

Get size of last part.

merge_parts(*version_id=None*, ***kwargs*)

Merge parts into object version.

classmethod query_by_bucket(*bucket*)

Query all uncompleted multipart uploads.

classmethod query_expired(*dt*, *bucket=None*)

Query all uncompleted multipart uploads.

size

Size of file.

updated

Modification timestamp.

upload_id

Identifier for the specific version of an object.

validate_key(*key*, *key_*)

Validate key.

class invenio_files_rest.models.**ObjectVersion**(***kwargs*)

Model for storing versions of objects.

A bucket stores one or more objects identified by a key. Each object is versioned where each version is represented by an `ObjectVersion`.

An object version can either be 1) a *normal version* which is linked to a file instance, or 2) a *delete marker*, which is *not* linked to a file instance.

An normal object version is linked to a physical file on disk via a file instance. This allows for multiple object versions to point to the same file on disk, to optimize storage efficiency (e.g. useful for snapshotting an entire bucket without duplicating the files).

A delete marker object version represents that the object at hand was deleted.

The latest version of an object is marked using the `is_head` property. If the latest object version is a delete marker the object will not be shown in the bucket.

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.

property basename

Return filename of the object.

bucket

Relationship to buckets.

bucket_id

Bucket identifier.

copy(*bucket=None*, *key=None*)

Copy an object version to a given bucket + object key.

The copy operation is handled completely at the metadata level. The actual data on disk is not copied. Instead, the two object versions will point to the same physical file (via the same FileInstance).

All the tags associated with the current object version are copied over to the new instance.

> **Warning:** If the destination object exists, it will be replaced by the new object version which will become the latest version.

Parameters

- **bucket** – The bucket (instance or id) to copy the object to. Default: current bucket.
- **key** – Key name of destination object. Default: current object key.

Returns

The copied object version.

classmethod create(*bucket*, *key*, *_file_id=None*, *stream=None*, *mimetype=None*, *version_id=None*, ***kwargs*)

Create a new object in a bucket.

The created object is by default created as a delete marker. You must use `set_contents()` or `set_location()` in order to change this.

Parameters

- **bucket** – The bucket (instance or id) to create the object in.
- **key** – Key of object.
- **_file_id** – For internal use.
- **stream** – File-like stream object. Used to set content of object immediately after being created.
- **mimetype** – MIME type of the file object if it is known.
- **kwargs** – Keyword arguments passed to `Object.set_contents()`.

created

Creation timestamp.

classmethod delete(*bucket*, *key*)

Delete an object.

Technically works by creating a new version which works as a delete marker.

Parameters

- **bucket** – The bucket (instance or id) to delete the object from.
- **key** – Key of object.

Returns
Created delete marker object if key exists else `None`.

`property deleted`
Determine if object version is a delete marker.

`file`
Relationship to file instance.

`file_id`
File instance for this object version.

A null value in this column defines that the object has been deleted.

`classmethod get`(*bucket*, *key*, *version_id=None*)
Fetch a specific object.

By default the latest object version is returned, if `version_id` is not set.

Parameters
- **bucket** – The bucket (instance or id) to get the object from.
- **key** – Key of object.
- **version_id** – Specific version of an object.

`classmethod get_by_bucket`(*bucket*, *versions=False*, *with_deleted=False*)
Return query that fetches all the objects in a bucket.

Parameters
- **bucket** – The bucket (instance or id) to query.
- **versions** – Select all versions if True, only heads otherwise.
- **with_deleted** – Select also deleted objects if True.

Returns
The query to retrieve filtered objects in the given bucket.

`get_tags`()
Get tags for object version as dictionary.

`classmethod get_versions`(*bucket*, *key*, *desc=True*)
Fetch all versions of a specific object.

Parameters
- **bucket** – The bucket (instance or id) to get the object from.
- **key** – Key of object.
- **desc** – Sort results desc if True, asc otherwise.

Returns
The query to execute to fetch all versions.

`is_head`
Defines if object is the latest version.

`classmethod ix_uq_partial_files_object_is_head_dll`()
Return DDL instruction for ix_uq_partial_files_object_is_head.

key

Key identifying the object.

mimetype

Get MIME type of object.

classmethod relink_all(*old_file*, *new_file*)

Relink all object versions (for a given file) to a new file.

> **Warning:** Use this method with great care.

remove()

Permanently remove a specific object version from the database.

> **Warning:** This by-passes the normal versioning and should only be used when you want to permanently delete a specific object version. Otherwise use *ObjectVersion.delete()*.
>
> Note the method does not remove the associated file instance which must be garbage collected.

Returns
`self`.

restore()

Restore this object version to become the latest version.

Raises an exception if the object is the latest version.

send_file(*restricted=True*, *trusted=False*, ***kwargs*)

Wrap around FileInstance's send file.

set_contents(*stream*, *chunk_size=None*, *size=None*, *size_limit=None*, *progress_callback=None*)

Save contents of stream to file instance.

If a file instance has already been set, this methods raises an `FileInstanceAlreadySetError` exception.

Parameters

- **stream** – File-like stream.
- **size** – Size of stream if known.
- **chunk_size** – Desired chunk size to read stream in. It is up to the storage interface if it respects this value.

set_file(*fileinstance*)

Set a file instance.

set_location(*uri*, *size*, *checksum*, *storage_class=None*)

Set only URI location of for object.

Useful to link files on externally controlled storage. If a file instance has already been set, this methods raises an `FileInstanceAlreadySetError` exception.

Parameters

- **uri** – Full URI to object (which can be interpreted by the storage interface).
- **size** – Size of file.

- **checksum** – Checksum of file.
- **storage_class** – Storage class where file is stored ()

updated

Modification timestamp.

validate_key(*key*, *key_*)

Validate key.

version_id

Identifier for the specific version of an object.

class invenio_files_rest.models.**ObjectVersionTag**(***kwargs*)

Model for storing tags associated to object versions.

Used for storing extra technical information for an object version.

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.

copy(*object_version=None*, *key=None*)

Copy a tag to a given object version.

Parameters

- **object_version** – The object version instance to copy the tag to. Default: current object version.
- **key** – Key of destination tag. Default: current tag key.

Returns

The copied object version tag.

classmethod create(*object_version*, *key*, *value*)

Create a new tag for a given object version.

classmethod create_or_update(*object_version*, *key*, *value*)

Create or update a new tag for a given object version.

classmethod delete(*object_version*, *key=None*)

Delete tags.

Parameters

- **object_version** – The object version instance or id.
- **key** – Key of the tag to delete. Default: delete all tags.

classmethod get(*object_version*, *key*)

Get the tag object.

classmethod get_value(*object_version*, *key*)

Get the tag value.

key

Tag key.

object_version

Relationship to object versions.

value

Tag value.

version_id

Object version id.

class invenio_files_rest.models.**Part**(***kwargs*)

Part object.

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.

checksum

String representing the checksum of the part.

classmethod count(*mp*)

Count number of parts for a given multipart object.

classmethod create(*mp*, *part_number*, *stream=None*, ***kwargs*)

Create a new part object in a multipart object.

created

Creation timestamp.

classmethod delete(*mp*, *part_number*)

Get part number.

property end_byte

Get end byte in file for this part.

classmethod get_or_create(*mp*, *part_number*)

Get or create a part.

classmethod get_or_none(*mp*, *part_number*)

Get part number.

multipart

Relationship to multipart objects.

part_number

Part number.

property part_size

Get size of this part.

classmethod query_by_multipart(*multipart*)

Get all parts for a specific multipart upload.

Parameters
multipart – A *invenio_files_rest.models.MultipartObject* instance.

Returns
A *invenio_files_rest.models.Part* instance.

set_contents(*stream*, *progress_callback=None*)

Save contents of stream to part of file instance.

If a the MultipartObject is completed this methods raises an `MultipartAlreadyCompleted` exception.

Parameters

- **stream** – File-like stream.
- **size** – Size of stream if known.
- **chunk_size** – Desired chunk size to read stream in. It is up to the storage interface if it respects this value.

property start_byte

Get start byte in file of this part.

updated

Modification timestamp.

upload_id

Multipart object identifier.

2.1.2 Storage

File storage interface.

class invenio_files_rest.storage.**FileStorage**(*size=None*, *modified=None*)

Base class for storage interface to a single file.

Initialize storage object.

checksum(*chunk_size=None*, *progress_callback=None*, ***kwargs*)

Compute checksum of file.

copy(*src*, *chunk_size=None*, *progress_callback=None*)

Copy data from another file instance.

Parameters

- **src** – Source stream.
- **chunk_size** – Chunk size to read from source stream.

delete()

Delete the file.

initialize(*size=0*)

Initialize the file on the storage + truncate to the given size.

open(*mode=None*)

Open the file.

The caller is responsible for closing the file.

save(*incoming_stream*, *size_limit=None*, *size=None*, *chunk_size=None*, *progress_callback=None*)

Save incoming stream to file storage.

send_file(*filename*, *mimetype=None*, *restricted=True*, *checksum=None*, *trusted=False*, *chunk_size=None*, *as_attachment=False*)

Send the file to the client.

update(*incoming_stream*, *seek=0*, *size=None*, *chunk_size=None*, *progress_callback=None*)

Update part of file with incoming stream.

class invenio_files_rest.storage.**PyFSFileStorage**(*fileurl*, *size=None*, *modified=None*, *clean_dir=True*)

File system storage using PyFilesystem for access the file.

This storage class will store files according to the following pattern: <base_uri>/<file instance uuid>/data.

Warning: File operations are not atomic. E.g. if errors happens during e.g. updating part of a file it will leave the file in an inconsistent state. The storage class tries as best as possible to handle errors and leave the system in a consistent state.

Storage initialization.

delete()

Delete a file.

The base directory is also removed, as it is assumed that only one file exists in the directory.

initialize(*size=0*)

Initialize file on storage and truncate to given size.

open(*mode='rb'*)

Open file.

The caller is responsible for closing the file.

save(*incoming_stream*, *size_limit=None*, *size=None*, *chunk_size=None*, *progress_callback=None*)

Save file in the file system.

update(*incoming_stream*, *seek=0*, *size=None*, *chunk_size=None*, *progress_callback=None*)

Update a file in the file system.

invenio_files_rest.storage.**pyfs_storage_factory**(*fileinstance=None*, *default_location=None*, *default_storage_class=None*, *filestorage_class=<class 'invenio_files_rest.storage.pyfs.PyFSFileStorage'>*, *fileurl=None*, *size=None*, *modified=None*, *clean_dir=True*)

Get factory function for creating a PyFS file storage instance.

2.1.3 Signals

Models for Invenio-Files-REST.

invenio_files_rest.signals.**file_deleted = <blinker.base.NamedSignal object at 0x7f1088b06810; 'file-deleted'>**

File deleted signal.

Sent when a file is deleted.

invenio_files_rest.signals.**file_downloaded = <blinker.base.NamedSignal object at 0x7f1088b06690; 'file-downloaded'>**

File downloaded signal.

Sent when a file is downloaded.

invenio_files_rest.signals.**file_uploaded = <blinker.base.NamedSignal object at 0x7f108b194b50; 'file-uploaded'>**

File uploaded signal.

Sent when a file is uploaded.

2.1.4 File streaming

File serving helpers for Files REST API.

invenio_files_rest.helpers.**MIMETYPE_WHITELIST = {'audio/mpeg', 'audio/ogg', 'audio/wav', 'audio/webm', 'image/gif', 'image/jpeg', 'image/png', 'image/tiff', 'text/plain'}**

List of whitelisted MIME types.

> **Warning:** Do not add new types to this list unless you know what you are doing. You could potentially open up for XSS attacks.

invenio_files_rest.helpers.**chunk_size_or_default**(*chunk_size*)

Use default chunksize if not configured.

invenio_files_rest.helpers.**compute_checksum**(*stream*, *algo*, *message_digest*, *chunk_size=None*, *progress_callback=None*)

Get helper method to compute checksum from a stream.

Parameters

- **stream** – File-like object.
- **algo** – Identifier for checksum algorithm.
- **messsage_digest** – A message digest instance.
- **chunk_size** – Read at most size bytes from the file at a time.
- **progress_callback** – Function accepting one argument with number of bytes read. (Default: `None`)

Returns

The checksum.

invenio_files_rest.helpers.**compute_md5_checksum**(*stream*, ***kwargs*)

Get helper method to compute MD5 checksum from a stream.

Parameters

stream – The input stream.

Returns

The MD5 checksum.

invenio_files_rest.helpers.**create_file_streaming_redirect_response**(*obj*)

Redirect response generating function.

invenio_files_rest.helpers.**make_path**(*base_uri*, *path*, *filename*, *path_dimensions*, *split_length*)

Generate a path as base location for file instance.

Parameters

- **base_uri** – The base URI.

- **path** – The relative path.
- **path_dimensions** – Number of chunks the path should be split into.
- **split_length** – The length of any chunk.

Returns
A string representing the full path.

invenio_files_rest.helpers.**populate_from_path**(*bucket*, *source*, *checksum=True*, *key_prefix=''*, *chunk_size=None*)

Populate a `bucket` from all files in path.

Parameters

- **bucket** – The bucket (instance or id) to create the object in.
- **source** – The file or directory path.
- **checksum** – If `True` then a MD5 checksum will be computed for each file. (Default: `True`)
- **key_prefix** – The key prefix for the bucket.
- **chunk_size** – Chunk size to read from file.

Returns
A iterator for all *invenio_files_rest.models.ObjectVersion* instances.

invenio_files_rest.helpers.**sanitize_mimetype**(*mimetype*, *filename=None*)

Sanitize a MIME type so the browser does not render the file.

invenio_files_rest.helpers.**send_stream**(*stream*, *filename*, *size*, *mtime*, *mimetype=None*, *restricted=True*, *as_attachment=False*, *etag=None*, *content_md5=None*, *chunk_size=None*, *conditional=True*, *trusted=False*)

Send the contents of a file to the client.

Warning: It is very easy to be exposed to Cross-Site Scripting (XSS) attacks if you serve user uploaded files. Here are some recommendations:

1. Serve user uploaded files from a separate domain (not a subdomain). This way a malicious file can only attack other user uploaded files.
2. Prevent the browser from rendering and executing HTML files (by setting `trusted=False`).
3. Force the browser to download the file as an attachment (`as_attachment=True`).

Parameters

- **stream** – The file stream to send.
- **filename** – The file name.
- **size** – The file size.
- **mtime** – A Unix timestamp that represents last modified time (UTC).
- **mimetype** – The file mimetype. If `None`, the module will try to guess. (Default: `None`)
- **restricted** – If the file is not restricted, the module will set the cache-control. (Default: `True`)
- **as_attachment** – If the file is an attachment. (Default: `False`)
- **etag** – If defined, it will be set as HTTP E-Tag.

- **content_md5** – If defined, a HTTP Content-MD5 header will be set.
- **chunk_size** – The chunk size.
- **conditional** – Make the response conditional to the request. (Default: `True`)
- **trusted** – Do not enable this option unless you know what you are doing. By default this function will send HTTP headers and MIME types that prevents your browser from rendering e.g. a HTML file which could contain a malicious script tag. (Default: `False`)

Returns
A Flask response instance.

2.1.5 Tasks

Celery tasks for Invenio-Files-REST.

`invenio_files_rest.tasks.`**clear_orphaned_files**(*force_delete_check=<function <lambda> at 0x7f1088aca830>*, *limit=1000*)

Delete orphaned files from DB and storage.

Note: Orphan files are files (*invenio_files_rest.models.FileInstance* objects and their on-disk counterparts) that do not have any *invenio_files_rest.models.ObjectVersion* objects associated with them (anymore).

The celery beat configuration for scheduling this task may set values for this task's parameters:

```
"clear-orphan-files": {
    "task": "invenio_files_rest.tasks.clear_orphaned_files",
    "schedule": 60 * 60 * 24,
    "kwargs": {
        "force_delete_check": lambda file: False,
        "limit": 500,
    }
}
```

Parameters

- **force_delete_check** – A function to be called on each orphan file instance to check if its deletion should be forced (bypass the check of its `writable` flag). For example, this function can be used to force-delete files only if they are located on the local file system. Signature: The function should accept a *invenio_files_rest.models.FileInstance* object and return a boolean value. Default: Never force-delete any orphan files (`lambda file_instance: False`).
- **limit** – Limit for the number of orphan files considered for deletion in each task execution (and thus the number of generated celery tasks). A value of zero (0) or lower disables the limit.

`invenio_files_rest.tasks.`**default_checksum_verification_files_query**()

Return a query of valid FileInstances for checksum verification.

`invenio_files_rest.tasks.`**merge_multipartobject**(*upload_id*, *version_id=None*)

Merge multipart object.

Parameters

- **upload_id** – The *invenio_files_rest.models.MultipartObject* upload ID.
- **version_id** – Optionally you can define which file version. (Default: None)

Returns
The *invenio_files_rest.models.ObjectVersion* version ID.

invenio_files_rest.tasks.**migrate_file**(*src_id*, *location_name*, *post_fixity_check=False*)

Task to migrate a file instance to a new location.

Note: If something goes wrong during the content copy, the destination file instance is removed.

Parameters

- **src_id** – The *invenio_files_rest.models.FileInstance* ID.
- **location_name** – Where to migrate the file.
- **post_fixity_check** – Verify checksum after migration. (Default: False)

invenio_files_rest.tasks.**progress_updater**(*size*, *total*)

Progress reporter for checksum verification.

invenio_files_rest.tasks.**remove_expired_multipartobjects**()

Remove expired multipart objects.

invenio_files_rest.tasks.**remove_file_data**(*file_id*, *silent=True*, *force=False*)

Remove file instance and associated data.

Parameters

- **file_id** – The *invenio_files_rest.models.FileInstance* ID.
- **silent** – It stops propagation of a possible raised IntegrityError exception. (Default: True)
- **force** – Whether to delete the file even if the file instance is not marked as writable.

Raises
sqlalchemy.exc.IntegrityError – Raised if the database removal goes wrong and silent is set to False.

invenio_files_rest.tasks.**schedule_checksum_verification**(*frequency=None*, *batch_interval=None*, *max_count=None*, *max_size=None*, *files_query=None*, *checksum_kwargs=None*)

Schedule a batch of files for checksum verification.

The purpose of this task is to be periodically called through *celerybeat*, in order achieve a repeated verification cycle of all file checksums, while following a set of constraints in order to throttle the execution rate of the checks.

Parameters

- **frequency** (*dict*) – Time period over which a full check of all files should be performed. The argument is a dictionary that will be passed as arguments to the *datetime.timedelta* class. Defaults to a month (30 days).
- **batch_interval** (*dict*) – How often a batch is sent. If not supplied, this information will be extracted, if possible, from the celery.conf['CELERYBEAT_SCHEDULE'] entry of this task. The argument is a dictionary that will be passed as arguments to the *datetime.timedelta* class.

- **max_count** (*int*) – Max count of files of a single batch. When set to *0* it's automatically calculated to be distributed equally through the number of total batches.
- **max_size** (*int*) – Max size of a single batch in bytes. When set to *0* it's automatically calculated to be distributed equally through the number of total batches.
- **files_query** (*str*) – Import path for a function returning a FileInstance query for files that should be checked.
- **checksum_kwargs** (*dict*) – Passed to `FileInstance.verify_checksum`.

invenio_files_rest.tasks.**verify_checksum**(*file_id*, *pessimistic=False*, *chunk_size=None*, *throws=True*, *checksum_kwargs=None*)

Verify checksum of a file instance.

Parameters
file_id – The file ID.

2.1.6 Exceptions

Errors for Invenio-Files-REST.

exception invenio_files_rest.errors.**BucketLockedError**(*errors=None*, ***kwargs*)

Exception raised when a bucket is locked.

Initialize RESTException.

exception invenio_files_rest.errors.**DuplicateTagError**(*errors=None*, ***kwargs*)

Invalid tag key and/or value.

Initialize RESTException.

exception invenio_files_rest.errors.**ExhaustedStreamError**(*errors=None*, ***kwargs*)

The incoming file stream has been already consumed.

Initialize RESTException.

exception invenio_files_rest.errors.**FileInstanceAlreadySetError**(*errors=None*, ***kwargs*)

Exception raised when file instance already set on object.

Initialize RESTException.

exception invenio_files_rest.errors.**FileInstanceUnreadableError**(*errors=None*, ***kwargs*)

Exception raised when trying to get an unreadable file.

Initialize RESTException.

exception invenio_files_rest.errors.**FileSizeError**(*errors=None*, ***kwargs*)

Exception raised when a file larger than allowed.

Initialize RESTException.

exception invenio_files_rest.errors.**FilesException**(*errors=None*, ***kwargs*)

Base exception for all errors.

Initialize RESTException.

exception invenio_files_rest.errors.**InvalidKeyError**(*errors=None*, ***kwargs*)

Invalid key.

Initialize RESTException.

exception invenio_files_rest.errors.**InvalidOperationError**(*errors=None*, ***kwargs*)
Exception raised when an invalid operation is performed.

Initialize RESTException.

exception invenio_files_rest.errors.**InvalidTagError**(*errors=None*, ***kwargs*)
Invalid tag key and/or value.

Initialize RESTException.

exception invenio_files_rest.errors.**MissingQueryParameter**(*arg_name*, ***kwargs*)
Exception raised when missing a query parameter.

Initialize RESTException.

get_description(*environ=None*)
Get the description.

exception invenio_files_rest.errors.**MultipartAlreadyCompleted**(*errors=None*, ***kwargs*)
Exception raised when multipart object is already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartException**(*errors=None*, ***kwargs*)
Exception for multipart objects.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartInvalidChunkSize**(*errors=None*, ***kwargs*)
Exception raised when multipart object is already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartInvalidPartNumber**(*errors=None*, ***kwargs*)
Exception raised when multipart object is already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartInvalidSize**(*errors=None*, ***kwargs*)
Exception raised when multipart object is already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartMissingParts**(*errors=None*, ***kwargs*)
Exception raised when multipart object is already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartNoPart**(*errors=None*, ***kwargs*)
Exception raised by part factories when no part was detected.

Initialize RESTException.

exception invenio_files_rest.errors.**MultipartNotCompleted**(*errors=None*, ***kwargs*)
Exception raised when multipart object is not already completed.

Initialize RESTException.

exception invenio_files_rest.errors.**StorageError**(*errors=None*, ***kwargs*)
Exception raised when a storage operation fails.

Initialize RESTException.

get_errors()

Get errors.

Returns

A string with the error message.

exception invenio_files_rest.errors.**UnexpectedFileSizeError**(*errors=None*, ***kwargs*)

Exception raised when a file does not match its expected size.

Initialize RESTException.

2.1.7 Limiters

File size limiting functionality for Invenio-Files-REST.

class invenio_files_rest.limiters.**FileSizeLimit**(*limit*, *reason*)

File size limiter.

Instantiate a new file size limit.

Parameters

- **limit** – The imposed imposed limit.
- **reason** – The limit description.

invenio_files_rest.limiters.**file_size_limiters**(*bucket*)

Get default file size limiters.

Parameters

bucket – The *invenio_files_rest.models.Bucket* instance.

Returns

A list containing an instance of *invenio_files_rest.limiters.FileSizeLimit* with quota left value and description and another one with max file size value and description.

2.1.8 Permissions

Permissions for files using Invenio-Access.

invenio_files_rest.permissions.**BucketListMultiparts** = **functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-bucket-listmultiparts')**

Action needed: list multipart uploads in bucket.

invenio_files_rest.permissions.**BucketRead** = **functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-bucket-read')**

Action needed: list objects in bucket.

invenio_files_rest.permissions.**BucketReadVersions** = **functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-bucket-read-versions')**

Action needed: list object versions in bucket.

invenio_files_rest.permissions.**BucketUpdate** = **functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-bucket-update')**

Action needed: create objects and multipart uploads in bucket.

invenio_files_rest.permissions.**LocationUpdate = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-location-update')**

Action needed: location update.

invenio_files_rest.permissions.**MultipartDelete = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-multipart-delete')**

Action needed: abort a multipart upload.

invenio_files_rest.permissions.**MultipartRead = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-multipart-read')**

Action needed: list parts of a multipart upload in a bucket.

invenio_files_rest.permissions.**ObjectDelete = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-object-delete')**

Action needed: delete object in bucket.

invenio_files_rest.permissions.**ObjectDeleteVersion = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-object-delete-version')**

Action needed: permanently delete specific object version in bucket.

invenio_files_rest.permissions.**ObjectRead = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-object-read')**

Action needed: get object in bucket.

invenio_files_rest.permissions.**ObjectReadVersion = functools.partial(functools.partial(<class 'invenio_access.permissions.Need'>, 'action'), 'files-rest-object-read-version')**

Action needed: get object version in bucket.

invenio_files_rest.permissions.**bucket_listmultiparts_all = Need(method='action', value='files-rest-bucket-listmultiparts', argument=None)**

Action needed: list all buckets multiparts.

invenio_files_rest.permissions.**bucket_read_all = Need(method='action', value='files-rest-bucket-read', argument=None)**

Action needed: read all buckets.

invenio_files_rest.permissions.**bucket_read_versions_all = Need(method='action', value='files-rest-bucket-read-versions', argument=None)**

Action needed: read all buckets versions.

invenio_files_rest.permissions.**bucket_update_all = Need(method='action', value='files-rest-bucket-update', argument=None)**

Action needed: update all buckets

invenio_files_rest.permissions.**location_update_all = Need(method='action', value='files-rest-location-update', argument=None)**

Action needed: update all locations.

invenio_files_rest.permissions.**multipart_delete_all = Need(method='action', value='files-rest-multipart-delete', argument=None)**

Action needed: delete all multiparts.

invenio_files_rest.permissions.**multipart_read_all = Need(method='action', value='files-rest-multipart-read', argument=None)**

Action needed: read all multiparts.

invenio_files_rest.permissions.**object_delete_all = Need(method='action', value='files-rest-object-delete', argument=None)**

Action needed: delete all objects.

invenio_files_rest.permissions.**object_delete_version_all = Need(method='action', value='files-rest-object-delete-version', argument=None)**

Action needed: delete all objects versions.

invenio_files_rest.permissions.**object_read_all = Need(method='action', value='files-rest-object-read', argument=None)**

Action needed: read all objects.

invenio_files_rest.permissions.**object_read_version_all = Need(method='action', value='files-rest-object-read-version', argument=None)**

Action needed: read all objects versions.

invenio_files_rest.permissions.**permission_factory**(*obj*, *action*)

Get default permission factory.

Parameters

- **obj** – An instance of *invenio_files_rest.models.Bucket* or *invenio_files_rest.models.ObjectVersion* or *invenio_files_rest.models.MultipartObject* or None if the action is global.
- **action** – The required action.

Raises

RuntimeError – If the object is unknown.

Returns

A invenio_access.permissions.Permission instance.

2.1.9 Views

Files download/upload REST API similar to S3 for Invenio.

class invenio_files_rest.views.**BucketResource**(**args*, ***kwargs*)

Bucket item resource.

Instantiate content negotiated view.

get(*bucket=None*, *versions=<marshmallow.missing>*, *uploads=<marshmallow.missing>*)

Get list of objects in the bucket.

Parameters

bucket – A *invenio_files_rest.models.Bucket* instance.

Returns

The Flask response.

head(*bucket=None*, ***kwargs*)

Check the existence of the bucket.

listobjects(*bucket*, *versions*)

List objects in a bucket.

Parameters
bucket – A *invenio_files_rest.models.Bucket* instance.

Returns
The Flask response.

methods: ClassVar[Optional[Collection[str]]] = {'GET', 'HEAD'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

multipart_listuploads(*bucket*)

List objects in a bucket.

Parameters
bucket – A *invenio_files_rest.models.Bucket* instance.

Returns
The Flask response.

class invenio_files_rest.views.**LocationResource**(**args*, ***kwargs*)

Service resource.

Instantiate content negotiated view.

methods: ClassVar[Optional[Collection[str]]] = {'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post()

Create bucket.

class invenio_files_rest.views.**ObjectResource**(**args*, ***kwargs*)

Object item resource.

Instantiate content negotiated view.

static check_object_permission(*obj*)

Retrieve object and abort if it doesn't exists.

create_object(*bucket*, *key*)

Create a new object.

Parameters

- **bucket** – The bucket (instance or id) to get the object from.
- **key** – The file key.

Returns
A Flask response.

delete(*bucket=None*, *key=None*, *version_id=None*, *upload_id=None*, *uploads=None*)

Delete an object or abort a multipart upload.

Parameters

- **bucket** – The bucket (instance or id) to get the object from. (Default: `None`)
- **key** – The file key. (Default: `None`)

- **version_id** – The version ID. (Default: None)
- **upload_id** – The upload ID. (Default: None)

Returns
A Flask response.

delete_object(*bucket*, *obj*, *version_id*)

Delete an existing object.

Parameters

- **bucket** – The bucket (instance or id) to get the object from.
- **obj** – A *invenio_files_rest.models.ObjectVersion* instance.
- **version_id** – The version ID.

Returns
A Flask response.

get(*bucket=None*, *key=None*, *version_id=None*, *upload_id=None*, *uploads=None*, *download=None*)

Get object or list parts of a multipart upload.

Parameters

- **bucket** – The bucket (instance or id) to get the object from. (Default: None)
- **key** – The file key. (Default: None)
- **version_id** – The version ID. (Default: None)
- **upload_id** – The upload ID. (Default: None)
- **download** – The download flag. (Default: None)

Returns
A Flask response.

classmethod get_object(*bucket*, *key*, *version_id*)

Retrieve object and abort if it doesn't exist.

If the file is not found, the connection is aborted and the 404 error is returned.

Parameters

- **bucket** – The bucket (instance or id) to get the object from.
- **key** – The file key.
- **version_id** – The version ID.

Returns
A *invenio_files_rest.models.ObjectVersion* instance.

methods: `ClassVar[Optional[Collection[str]]] = {'DELETE', 'GET', 'POST', 'PUT'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

multipart_complete(*multipart*)

Complete a multipart upload.

Parameters
multipart – A *invenio_files_rest.models.MultipartObject* instance.

Returns
A Flask response.

`multipart_delete`(*multipart*)

Abort a multipart upload.

Parameters
multipart – A *invenio_files_rest.models.MultipartObject* instance.

Returns
A Flask response.

`multipart_init`(*bucket*, *key*, *size=None*, *part_size=None*)

Initialize a multipart upload.

Parameters

- **bucket** – The bucket (instance or id) to get the object from.
- **key** – The file key.
- **size** – The total size.
- **part_size** – The part size.

Raises
invenio_files_rest.errors.MissingQueryParameter – If size or part_size are not defined.

Returns
A Flask response.

`multipart_listparts`(*multipart*)

Get parts of a multipart upload.

Parameters
multipart – A *invenio_files_rest.models.MultipartObject* instance.

Returns
A Flask response.

`multipart_uploadpart`(*multipart*)

Upload a part.

Parameters
multipart – A *invenio_files_rest.models.MultipartObject* instance.

Returns
A Flask response.

`post`(*bucket=None*, *key=None*, *uploads=<marshmallow.missing>*, *upload_id=None*)

Upload a new object or start/complete a multipart upload.

Parameters

- **bucket** – The bucket (instance or id) to get the object from. (Default: `None`)
- **key** – The file key. (Default: `None`)
- **upload_id** – The upload ID. (Default: `None`)

Returns
A Flask response.

put(*bucket=None*, *key=None*, *upload_id=None*)

Update a new object or upload a part of a multipart upload.

Parameters

- **bucket** – The bucket (instance or id) to get the object from. (Default: `None`)
- **key** – The file key. (Default: `None`)
- **upload_id** – The upload ID. (Default: `None`)

Returns
A Flask response.

static send_object(*bucket*, *obj*, *expected_chksum=None*, *logger_data=None*, *restricted=True*, *as_attachment=False*)

Send an object for a given bucket.

Parameters

- **bucket** – The bucket (instance or id) to get the object from.
- **obj** – A *invenio_files_rest.models.ObjectVersion* instance.
- **logger_data** – The python logger.
- **kwargs** – Keyword arguments passed to `Object.send_file()`

Params expected_chksum
Expected checksum.

Returns
A Flask response.

invenio_files_rest.views.**as_uuid**(*value*)

Convert value to UUID.

invenio_files_rest.views.**bucket_view**(***kwargs*)

Bucket item resource.

Parameters
kwargs (*Any*) –

Return type
Union[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int, *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]], WSGIApplication]

invenio_files_rest.views.**check_permission**(*permission*, *hidden=True*)

Check if permission is allowed.

If permission fails then the connection is aborted.

Parameters

- **permission** – The permission to check.
- **hidden** – Determine if a 404 error (`True`) or 401/403 error (`False`) should be returned if the permission is rejected (i.e. hide or reveal the existence of a particular object).

invenio_files_rest.views.**default_partfactory**(*part_number=None*, *content_length=None*, *content_type=None*, *content_md5=None*)

Get default part factory.

Parameters

- **part_number** – The part number. (Default: None)
- **content_length** – The content length. (Default: None)
- **content_type** – The HTTP Content-Type. (Default: None)
- **content_md5** – The content MD5. (Default: None)

Returns
The content length, the part number, the stream, the content type, MD5 of the content.

invenio_files_rest.views.**ensure_input_stream_is_not_exhausted**(*f*)

Make sure that the input stream has not been read already.

invenio_files_rest.views.**invalid_subresource_validator**(*value*)

Ensure subresource.

invenio_files_rest.views.**location_view**(***kwargs*)

Service resource.

Parameters
kwargs (*Any*) –

Return type
Union[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int, *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]], WSGIApplication]

invenio_files_rest.views.**minsize_validator**(*value*)

Validate Content-Length header.

Raises
invenio_files_rest.errors.FileSizeError – If the value is less than *invenio_files_rest.config.FILES_REST_MIN_FILE_SIZE* size.

invenio_files_rest.views.**need_bucket_permission**(*action*, *hidden=True*)

Get permission for buckets or abort.

Parameters

- **object_getter** – The function used to retrieve the object and pass it to the permission factory.
- **action** – The action needed.
- **hidden** – Determine which kind of error to return. (Default: True)

invenio_files_rest.views.**need_location_permission**(*action*, *hidden=True*)

Get permission for buckets or abort.

Parameters

- **object_getter** – The function used to retrieve the object and pass it to the permission factory.
- **action** – The action needed.
- **hidden** – Determine which kind of error to return. (Default: `True`)

invenio_files_rest.views.**need_permissions**(*object_getter*, *action*, *hidden=True*)

Get permission for buckets or abort.

Parameters

- **object_getter** – The function used to retrieve the object and pass it to the permission factory.
- **action** – The action needed.
- **hidden** – Determine which kind of error to return. (Default: `True`)

invenio_files_rest.views.**ngfileupload_partfactory**(*part_number=None*, *content_length=None*, *uploaded_file=None*)

Part factory for ng-file-upload.

Parameters

- **part_number** – The part number. (Default: `None`)
- **content_length** – The content length. (Default: `None`)
- **uploaded_file** – The upload request. (Default: `None`)

Returns

The content length, part number, stream, HTTP Content-Type header.

invenio_files_rest.views.**ngfileupload_uploadfactory**(*content_length=None*, *content_type=None*, *uploaded_file=None*)

Get default put factory.

If Content-Type is `'multipart/form-data'` then the stream is aborted.

Parameters

- **content_length** – The content length. (Default: `None`)
- **content_type** – The HTTP Content-Type. (Default: `None`)
- **uploaded_file** – The upload request. (Default: `None`)
- **file_tags_header** – The file tags. (Default: `None`)

Returns

A tuple containing stream, content length, and empty header.

invenio_files_rest.views.**object_view**(***kwargs*)

Object item resource.

Parameters

kwargs (*Any*) –

Return type

Union[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*], *Mapping*[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int], *Tuple*[*Union*[Response, str, bytes, *List*[*Any*],

Mapping[str, *Any*], *Iterator*[str], *Iterator*[bytes]], int, *Union*[Headers, *Mapping*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]], *Sequence*[*Tuple*[str, *Union*[str, *List*[str], *Tuple*[str, ...]]]]]], WSGIApplication]

invenio_files_rest.views.**parse_header_tags**()

Parse tags specified in the HTTP request header.

invenio_files_rest.views.**pass_bucket**(*f*)

Decorate to retrieve a bucket.

invenio_files_rest.views.**pass_multipart**(*with_completed=False*)

Decorate to retrieve an object.

invenio_files_rest.views.**stream_uploadfactory**(*content_md5=None*, *content_length=None*, *content_type=None*)

Get default put factory.

If Content-Type is `'multipart/form-data'` then the stream is aborted.

Parameters

- **content_md5** – The content MD5. (Default: `None`)
- **content_length** – The content length. (Default: `None`)
- **content_type** – The HTTP Content-Type. (Default: `None`)

Returns

The stream, content length, MD5 of the content.

invenio_files_rest.views.**validate_tag**(*key*, *value*)

Validate a tag.

Keys must be less than 128 chars and values must be less than 256 chars.

2.1.10 Form parser

Werkzeug form data parser customization.

class invenio_files_rest.formparser.**FormDataParser**(*stream_factory=None*, *charset='utf-8'*, *errors='replace'*, *max_form_memory_size=None*, *max_content_length=None*, *cls=None*, *silent=True*, ***, *max_form_parts=None*)

Custom form data parser.

Parameters

- **stream_factory** (*Optional[TStreamFactory]*) –
- **charset** (*str*) –
- **errors** (*str*) –
- **max_form_memory_size** (*Optional[int]*) –
- **max_content_length** (*Optional[int]*) –
- **cls** (*Optional[Type[MultiDict]]*) –
- **silent** (*bool*) –
- **max_form_parts** (*Optional[int]*) –

parse(*stream*, *mimetype*, *content_length*, *options=None*)

Parse the information from the given request.

Parameters

- **stream** – An input stream.
- **mimetype** – The mimetype of the data.
- **content_length** – The content length of the incoming data.
- **options** – Optional mimetype parameters (used for the multipart boundary for instance).

Returns

A tuple in the form `(stream, form, files)`.

CHAPTER

THREE

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-files-rest/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-Files-REST could always use more documentation, whether as part of the official Invenio-Files-REST 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-files-rest/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:inveniosoftware/invenio-files-rest.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-files-rest
$ cd invenio-files-rest/
$ 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 3.6, 3.7, 3.8 and 3.9. Check https://github.com/inveniosoftware/invenio-files-rest/actions?query=event%3Apull_request and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.5.0 (release 2023-03-02)

- remove deprecated flask-babelex dependency and imports
- install invenio-i18n

Version 1.4.0 (release 2023-01-24)

- tasks: add orphan cleaning celery task

Version 1.3.3 (release 2022-04-06)

- Fix Flask v2.1 issues.
- Refactor dependencies to respect Invenio dependency strategy and remove pin on Flask-Login.

Version 1.3.2 (release 2022-02-14)

- Fix deprecation warnings from marshmallow.

Version 1.3.1 (release 2022-01-31)

- Fix a race-condition by enforcing integrity constraint on is head. An issue was detected that could produce two head versions of the same object. This fix adds a partial index in PostgreSQL to ensure that the race condition throws an integrity error when trying to commit. Partial indexes is only available on PostgreSQL.
- Fix for the sync method and signals signature.

Version 1.3.0 (released 2021-10-18)

- Bumped minimum PyFilesystem dependency to v2. Note that, setuptools v58+ have dropped support for use2to3, thus PyFilesystem v0.5.5 no longer installs on Python 3 when using setuptools v58 or greater.

Version 1.2.0 (released 2020-05-14)

- Adds optional file streaming using a reverse proxy (e.g. NGINX).

Version 1.1.1 (released 2020-02-24)

- Makes cli *location* command backwards compatible.

Version 1.1.0 (released 2020-01-19)

- Moves *location* from command to group
- Allows listing locations via de CLI
- Allows setting a location as *default*
- Get by name on the *Location* object returns None when not found instead of raising an exception

- Other bug fixes

Version 1.0.6 (released 2019-11-22)

- Bump version and add to installation requirements invenio-celery
- Add documentation of module usage
- Remove storage_class parameter from Bucket create when POST to Location resource

Version 1.0.5 (released 2019-11-21)

- Add signals for deletion and upload of files

Version 1.0.4 (released 2019-11-20)

- Fix *StorageError* type returned

Version 1.0.3 (released 2019-11-15)

- Increase invenio-rest version to support Marshmallow 2 and 3 migration

Version 1.0.2 (released 2019-11-14)

- Adds optional serializer_mapping and view_name in *json_serializer* method

Version 1.0.1 (released 2019-08-01)

- Adds support for marshmallow 2 and 3.

Version 1.0.0 (released 2019-07-22)

- Initial public release.

3.3 License

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Chiara Bigarella
- David Zerulla
- Emanuel Dima
- Esteban J. G. Gabancho
- Harris Tzovanakis
- Ioan Ungurean
- Jacopo Notarstefano
- Javier Delgado
- Javier Martin Montull
- Jiri Kuncar
- Jose Benito Gonzalez Lopez
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Nicola Tarocco
- Nicolas Harraudeau
- Niklas Persson
- Nikos Filippakis
- Sami Hiltunen
- Samuele Kaplun
- Sebastian Witowski
- Spiros Delviniotis
- Steven Loria
- Tibor Simko

PYTHON MODULE INDEX

i

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

U

V

W