invenio-deposit Documentation

Release 1.0.0a9

CERN

Dec 06, 2017

Contents

Module for depositing record metadata and uploading files.

This is an experimental developer preview release.

- Free software: GPLv2 license
- Documentation: https://invenio-deposit.readthedocs.io/

CHAPTER 1

User's Guide

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

1.1 Installation

1.2 Usage

Module for depositing record metadata and uploading files.

CHAPTER 2

API Reference

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

2.1 API Docs

Module for depositing record metadata and uploading files.

class invenio_deposit.ext.**InvenioDeposit**(*app=None*)
Invenio-Deposit extension.

Extension initialization.

init_app(*app*)
Flask application initialization.

Initialize the UI endpoints. Connect all signals if *DEPOSIT_REGISTER_SIGNALS* is True.

Parameters **app** – An instance of flask.Flask.

init_config(*app*)
Initialize configuration.

Parameters **app** – An instance of flask.Flask.

class invenio_deposit.ext.**InvenioDepositREST**(*app=None*)
Invenio-Deposit REST extension.

Extension initialization.

Parameters **app** – An instance of flask.Flask.

init_app(*app*)
Flask application initialization.

Initialize the REST endpoints. Connect all signals if *DEPOSIT_REGISTER_SIGNALS* is True.

Parameters **app** – An instance of flask.Flask.

init_config(*app*)
Initialize configuration.

Parameters **app** – An instance of `flask.Flask`.

2.1.1 API

Deposit API.

class `invenio_deposit.api.`**Deposit**(*data*, *model=None*)
Define API for changing deposit state.

Initialize instance with dictionary data and SQLAlchemy model.

Parameters

- **data** – Dict with record metadata.
- **model** – `RecordMetadata` instance.

build_deposit_schema(*record*)
Convert record schema to a valid deposit schema.

Parameters **record** – The record used to build deposit schema.

Returns The absolute URL to the schema or *None*.

clear(**args*, ***kwargs*)
Clear only drafts.

Status required: `'draft'`.

Meta information inside *_deposit* are preserved.

commit(*self_or_cls*, **args*, ***kwargs*)
Store changes on current instance in database and index it.

classmethod **create**(*self_or_cls*, **args*, ***kwargs*)
Create a deposit.

Initialize the follow information inside the deposit:

```
deposit['_deposit'] = {
    'id': pid_value,
    'status': 'draft',
    'owners': [user_id],
    'created_by': user_id,
}
```

The deposit index is updated.

Parameters

- **data** – Input dictionary to fill the deposit.
- **id** – Default uuid for the deposit.

Returns The new created deposit.

delete(**args*, ***kwargs*)
Delete deposit.

Status required: `'draft'`.

Parameters

- **force** – Force deposit delete. (Default: `True`)
- **pid** – Force pid object. (Default: `None`)

Returns A new Deposit object.

static deposit_fetcher(*record_uuid*, *data*)
Function used to retrieve the deposit PID.

static deposit_minter(*record_uuid*, *data*)
Function used to mint the deposit PID.

discard(**args*, ***kwargs*)
Discard deposit changes.

1. **The signal invenio_records.signals.before_record_update is** sent before the edit execution.
2. It restores the last published version.
3. The following meta information are saved inside the deposit:

```
deposit['$schema'] = deposit_schema_from_record_schema
```

1. **The signal invenio_records.signals.after_record_update is** sent after the edit execution.
2. The deposit index is updated.

Status required: `'draft'`.

Parameters **pid** – Force a pid object. (Default: `None`)

Returns A new Deposit object.

edit(**args*, ***kwargs*)
Edit deposit.

1. The signal `invenio_records.signals.before_record_update` is sent before the edit execution.
2. The following meta information are saved inside the deposit:

```
deposit['_deposit']['pid'] = record.revision_id
deposit['_deposit']['status'] = 'draft'
deposit['$schema'] = deposit_schema_from_record_schema
```

1. **The signal invenio_records.signals.after_record_update is** sent after the edit execution.
2. The deposit index is updated.

Status required: *published*.

Note: the process fails if the pid has status `invenio_pidstore.models.PIDStatus.REGISTERED`.

Parameters **pid** – Force a pid object. (Default: `None`)

Returns A new Deposit object.

fetch_published()
Return a tuple with PID and published record.

files
List of Files inside the deposit.

Add validation on `sort_by` method: if, at the time of files access, the record is not a `'draft'` then a `invenio_pidstore.errors.PIDInvalidAction` is rised.

indexer = <invenio_indexer.api.RecordIndexer object>
Default deposit indexer.

merge_with_published(**args*, ***kwargs*)
Merge changes with latest published version.

patch(**args*, ***kwargs*)
Patch only drafts.

Status required: `'draft'`.

Meta information inside *_deposit* are preserved.

pid
Return an instance of deposit PID.

publish(**args*, ***kwargs*)
Publish a deposit.

If it's the first time:

- **it calls the minter and set the following meta information inside** the deposit:

```
deposit['_deposit'] = {
    'type': pid_type,
    'value': pid_value,
    'revision_id': 0,
}
```

- A dump of all information inside the deposit is done.
- A snapshot of the files is done.

Otherwise, published the new edited version. In this case, if in the mainwhile someone already published a new version, it'll try to merge the changes with the latest version.

Note: no need for indexing as it calls *self.commit()*.

Status required: `'draft'`.

Parameters

- **pid** – Force the new pid value. (Default: `None`)
- **id** – Force the new uuid value as deposit id. (Default: `None`)

Returns Returns itself.

published_record_class
The Record API class used for published records.

alias of `Record`

record_schema
Convert deposit schema to a valid record schema.

status
Property for accessing deposit status.

update(**args*, ***kwargs*)
Update only drafts.

Status required: `'draft'`.

Meta information inside *_deposit* are preserved.

invenio_deposit.api.**has_status**(*method=None*, *status='draft'*)
Check that deposit has a defined status (default: draft).

Parameters

- **method** – Function executed if record has a defined status. (Default: `None`)
- **status** – Defined status to check. (Default: `'draft'`)

invenio_deposit.api.**index**(*method=None*, *delete=False*)
Decorator to update index.

Parameters

- **method** – Function wrapped. (Default: `None`)
- **delete** – If *True* delete the indexed record. (Default: `None`)

invenio_deposit.api.**preserve**(*method=None*, *result=True*, *fields=None*)
Preserve fields in deposit.

Parameters

- **method** – Function to execute. (Default: `None`)
- **result** – If *True* returns the result of method execution, otherwise *self*. (Default: `True`)
- **fields** – List of fields to preserve (default: `('_deposit',)`).

Persistent identifier fetchers.

invenio_deposit.fetchers.**deposit_fetcher**(*record_uuid*, *data*)
Fetch a deposit identifier.

Parameters

- **record_uuid** – Record UUID.
- **data** – Record content.

Returns A `invenio_pidstore.fetchers.FetchedPID` that contains data['_deposit']['id'] as pid_value.

Persistent identifier minters.

invenio_deposit.minters.**deposit_minter**(*record_uuid*, *data*)
Mint a deposit identifier.

A PID with the following characteristics is created:

```
{
    "object_type": "rec",
    "object_uuid": record_uuid,
    "pid_value": "<new-pid-value>",
```

```
    "pid_type": "depid",
}
```

The following deposit meta information are updated:

```
deposit['_deposit'] = {
    "id": "<new-pid-value>",
    "status": "draft",
}
```

Parameters

- **record_uuid** – Record UUID.
- **data** – Record content.

Returns A invenio_pidstore.models.PersistentIdentifier object.

Deposit identifier provider.

class invenio_deposit.providers.**DepositProvider**(*pid*)

Deposit identifier provider.

Initialize provider using persistent identifier.

Parameters **pid** – A invenio_pidstore.models.PersistentIdentifier instance.

classmethod create(*object_type=None*, *object_uuid=None*, ***kwargs*)

Create a new deposit identifier.

Parameters

- **object_type** – The object type (Default: None)
- **object_uuid** – The object UUID (Default: None)
- **kwargs** – It contains the pid value.

default_status = 'R'

Deposit IDs are by default registered immediately.

pid_provider = None

Provider name.

The provider name is not recorded in the PID since the provider does not provide any additional features besides creation of deposit ids.

pid_type = 'depid'

Type of persistent identifier.

Deposit listeners.

invenio_deposit.receivers.**index_deposit_after_publish**(*sender*, *action=None*, *pid=None*, *deposit=None*)

Index the record after publishing.

Note: if the record is not published, it doesn't index.

Parameters

- **sender** – Who send the signal.

- **action** – Action executed by the sender. (Default: None)
- **pid** – PID object. (Default: None)
- **deposit** – Deposit object. (Default: None)

2.1.2 Configuration

Default configuration of deposit module.

invenio_deposit.config.**DEPOSIT_DEFAULT_JSONSCHEMA = 'deposits/deposit-v1.0.0.json'**
Default JSON schema used for new deposits.

invenio_deposit.config.**DEPOSIT_DEFAULT_SCHEMAFORM = 'json/invenio_deposit/form.json'**
Default Angular Schema Form.

invenio_deposit.config.**DEPOSIT_DEFAULT_STORAGE_CLASS = 'S'**
Default storage class.

invenio_deposit.config.**DEPOSIT_FILES_API = '/api/files'**
URL of files endpoints for uploading.

invenio_deposit.config.**DEPOSIT_FORM_TEMPLATES = {'fieldset': 'fieldset.html', 'textarea': 'textarea.html', 'radios'**
Templates for Angular Schema Form.

invenio_deposit.config.**DEPOSIT_FORM_TEMPLATES_BASE = 'node_modules/invenio-records-js/dist/templates'**
Angular Schema Form temmplates location.

invenio_deposit.config.**DEPOSIT_JSONSCHEMAS_PREFIX = 'deposits/'**
Prefix for all deposit JSON schemas.

invenio_deposit.config.**DEPOSIT_PID_MINTER = 'recid'**
PID minter used for record submissions.

invenio_deposit.config.**DEPOSIT_RECORDS_API = '/api/deposits/{pid_value}'**
URL of record endpoint for deposits.

invenio_deposit.config.**DEPOSIT_RECORDS_UI_ENDPOINTS = {'depid': {'record_class': 'invenio_deposit.api:Depos**
Basic deposit UI endpoints configuration.

The structure of the dictionary is as follows:

```
DEPOSIT_RECORDS_UI_ENDPOINTS = {
    '<pid-type>': {
        'pid_type': '<pid-type>',
        'route': '/unique/path/to/deposit/<pid_value>',
        'template': 'invenio_deposit/edit.html',
        'record_class': 'mypackage.api:MyDeposit',
        'view_imp': 'mypackage.views.view_method',
        'jsonschema' 'path/to/jsonschema/deposit.json',
        'schemaform': 'path/to/schema/form.json',
    }
}
```

invenio_deposit.config.**DEPOSIT_REGISTER_SIGNALS = True**
Enable the signals registration.

invenio_deposit.config.**DEPOSIT_RESPONSE_MESSAGES = {}**
Alerts shown when actions are completed on deposit.

invenio_deposit.config.**DEPOSIT_REST_DEFAULT_SORT = {'deposits': {'noquery': 'mostrecent', 'query': 'bestmatc**
Default deposit sort configuration. See invenio_records_rest.config.RECORDS_REST_DEFAULT_SORT for more information.

invenio_deposit.config.**DEPOSIT_REST_ENDPOINTS = {'depid': {'list_route': '/deposits/', 'indexer_class': None, 'de**
Basic REST deposit configuration.

Most of the configurations have the same meaning of the record configuration invenio_records_rest.config.RECORDS_REST_ENDPOINTS. Deposit introduce also configuration for files.

invenio_deposit.config.**DEPOSIT_REST_FACETS = {'deposits': {'post_filters': {'status': <function inner>}, 'aggs': {'s**
Basic deposit facts configuration. See invenio_records_rest.config.RECORDS_REST_FACETS for more information.

invenio_deposit.config.**DEPOSIT_REST_SORT_OPTIONS = {'deposits': {'bestmatch': {'fields': ['-_score'], 'default_o**
Basic deposit sort configuration. See invenio_records_rest.config.RECORDS_REST_SORT_OPTIONS for more information.

invenio_deposit.config.**DEPOSIT_SEARCH_API = '/api/deposits'**
URL of search endpoint for deposits.

invenio_deposit.config.**DEPOSIT_UI_ENDPOINT = '{scheme}://{host}/deposit/{pid_value}'**
The UI endpoint for depositions with pid.

invenio_deposit.config.**DEPOSIT_UI_INDEX_TEMPLATE = 'invenio_deposit/index.html'**
Template for the index page.

invenio_deposit.config.**DEPOSIT_UI_JSTEMPLATE_ACTIONS = 'node_modules/invenio-records-js/dist/templates/act**
Template for <invenio-records-actions> defined by *invenio-records-js*.

invenio_deposit.config.**DEPOSIT_UI_JSTEMPLATE_ERROR = 'node_modules/invenio-records-js/dist/templates/error**
Template for <invenio-records-error> defined by *invenio-records-js*.

invenio_deposit.config.**DEPOSIT_UI_JSTEMPLATE_FORM = 'node_modules/invenio-records-js/dist/templates/form.h**
Template for <invenio-records-form> defined by *invenio-records-js*.

invenio_deposit.config.**DEPOSIT_UI_NEW_TEMPLATE = 'invenio_deposit/edit.html'**
Template for a new deposit page.

invenio_deposit.config.**DEPOSIT_UI_SEARCH_INDEX = 'deposits'**
Search index name for the deposit.

invenio_deposit.config.**DEPOSIT_UI_TOMBSTONE_TEMPLATE = 'invenio_deposit/tombstone.html'**
Template for a tombstone deposit page.

Permissions for deposit.

invenio_deposit.permissions.**admin_permission_factory**()
Factory for creating a permission for an admin *deposit-admin-access*.

If *invenio-access* module is installed, it returns a invenio_access.permissions.DynamicPermission object. Otherwise, it returns a flask_principal.Permission object.

Returns Permission instance.

Deposit module signals.

invenio_deposit.signals.**post_action = <blinker.base.NamedSignal object at 0x7fad6bb753d0; 'post-action'>**
Signal is sent after the REST action.

Kwargs:

1. action (str) - name of REST action, e.g. "publish".

2. **pid (invenio_pidstore.models.PersistentIdentifier) - PID of the deposit.** The pid_type is assumed to be 'depid'.
3. deposit (invenio_depost.api.Deposit) - API instance of the deposit

Example subscriber:

```
def listener(sender, action, pid, deposit):
    pass

from invenio_deposit.signals import post_action
post_action.connect(listener)
```

Implementention of various utility functions.

invenio_deposit.utils.**can_elasticsearch**(*record*)
Check if a given record is indexed.

Parameters **record** – A record object.

Returns If the record is indexed returns *True*, otherwise *False*.

invenio_deposit.utils.**check_oauth2_scope**(*can_method*, **myscopes*)
Base permission factory that check OAuth2 scope and can_method.

Parameters

- **can_method** – Permission check function that accept a record in input and return a boolean.
- **myscopes** – List of scopes required to permit the access.

Returns A `flask_principal.Permission` factory.

invenio_deposit.utils.**check_oauth2_scope_write**(*record*, **args*, ***kwargs*)
Permission factory that check oauth2 scope.

The scope *invenio_deposit.scopes.write_scope* is checked.

invenio_deposit.utils.**check_oauth2_scope_write_elasticsearch**(*record*, **args*, ***kwargs*)
Permission factory that check oauth2 scope and if the record is indexed.

The scope *invenio_deposit.scopes.write_scope* is checked.

invenio_deposit.utils.**extract_actions_from_class**(*record_class*)
Extract actions from class.

invenio_deposit.utils.**mark_as_action**(*f*)
Decorator for marking method as deposit action.

Allows creation of new REST API action on `Deposit` subclass. Following example shows possible *cloning* of a deposit instance.

```
from invenio_deposit.api import Deposit

class CustomDeposit(Deposit):
    @mark_as_action
    def clone(self, pid=None):
        new_bucket = self.files.bucket.clone()
        new_deposit = Deposit.create(self.dumps())
        new_deposit.files.bucket = new_bucket
        new_deposit.commit()
```

```
    @mark_as_action
    def edit(self, pid=None):
        # Extend existing action.
        self['_last_editor'] = current_user.get_id()
        return super(CustomDeposit, self).edit(pid=pid)

    # Disable publish action from REST API.
    def publish(self, pid=None):
        return super(CustomDeposit, self).publish(pid=pid)
```

Parameters **f** – Decorated method.

2.1.3 Views

Links for record serialization.

invenio_deposit.links.**deposit_links_factory**(*pid*)
Factory for record links generation.

The dictionary is formed as:

```
{
    'files': '/url/to/files',
    'publish': '/url/to/publish',
    'edit': '/url/to/edit',
    'discard': '/url/to/discard',
    ...
}
```

Parameters **pid** – The record PID object.

Returns A dictionary that contains all the links.

OAuth2 scopes.

class invenio_deposit.scopes.**DepositScope**(*id_*, **args*, ***kwargs*)
Basic deposit scope.

Define the scope.

invenio_deposit.scopes.**actions_scope = <invenio_deposit.scopes.DepositScope object>**
Allow publishing of uploads.

invenio_deposit.scopes.**write_scope = <invenio_deposit.scopes.DepositScope object>**
Allow upload (but not publishing).

Configuration for deposit search.

class invenio_deposit.search.**DepositSearch**(***kwargs*)
Default search class.

Use Meta to set kwargs defaults.

class Meta
Configuration for deposit search.

invenio_deposit.search.**deposits_filter**()
Filter list of deposits.

Permit to the user to see all if:

- **The user is an admin (see** func:*invenio_deposit.permissions:admin_permission_factory*).
- It's called outside of a request.

Otherwise, it filters out any deposit where user is not the owner.

Deposit serializers.

invenio_deposit.serializers.**file_serializer**(*obj*)
Serialize a object.

Parameters **obj** – A invenio_files_rest.models.ObjectVersion instance.

Returns A dictionary with the fields to serialize.

invenio_deposit.serializers.**json_file_response**(*obj=None*, *pid=None*, *record=None*, *status=None*)
JSON Files/File serializer.

Parameters

- **obj** – A invenio_files_rest.models.ObjectVersion instance or a invenio_records_files.api.FilesIterator if it's a list of files.
- **pid** – PID value. (not used)
- **record** – The record metadata. (not used)
- **status** – The HTTP status code.

Returns A Flask response with JSON data.

Return type flask.Response.

invenio_deposit.serializers.**json_file_serializer**(*obj*, *status=None*)
JSON File Serializer.

Parameters

- **obj** – A invenio_files_rest.models.ObjectVersion instance.
- **status** – A HTTP Status. (Default: None)

Returns A Flask response with JSON data.

Return type flask.Response.

invenio_deposit.serializers.**json_files_serializer**(*objs*, *status=None*)
JSON Files Serializer.

Parma objs A list of:class:*invenio_files_rest.models.ObjectVersion* instances.

Parameters **status** – A HTTP Status. (Default: None)

Returns A Flask response with JSON data.

Return type flask.Response.

invenio_deposit.serializers.**json_serializer**(*pid*, *data*, **args*)
Build a JSON Flask response using the given data.

Parameters

- **pid** – The *invenio_pidstore.models.PersistentIdentifier* of the record.
- **data** – The record metadata.

Returns A Flask response with JSON data.

Return type flask.Response.

invenio_deposit.serializers.**json_v1_files_response**(*obj=None*, *pid=None*, *record=None*, *status=None*)

Default JSON files response.

Deposit actions.

class invenio_deposit.views.rest.**DepositActionResource**(*serializers*, *pid_type*, *ctx*, **args*, ***kwargs*)

Deposit action resource.

Constructor.

post(*pid_value*, **args*, ***kwargs*)

Handle deposit action.

After the action is executed, a *invenio_deposit.signals.post_action* signal is sent.

Permission required: *update_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.
- **action** – The action to execute.

class invenio_deposit.views.rest.**DepositFileResource**(*serializers*, *pid_type*, *ctx*, **args*, ***kwargs*)

Deposit files resource.

Constructor.

delete(**args*, ***kwargs*)

Handle DELETE deposit file.

Permission required: *update_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.
- **key** – Unique identifier for the file in the deposit.

get(**args*, ***kwargs*)

Get file.

Permission required: *read_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.
- **key** – Unique identifier for the file in the deposit.
- **version_id** – File version. Optional. If no version is provided, the last version is retrieved.

Returns the file content.

get_args = {'version_id': <fields.UUID(default=<marshmallow.missing>, attribute=None, validate=None, required=Fal

GET query arguments.

put (**args*, ***kwargs*)
Handle the file rename through the PUT deposit file.

Permission required: *update_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.
- **key** – Unique identifier for the file in the deposit.

class invenio_deposit.views.rest.**DepositFilesResource**(*serializers*, *pid_type*, *ctx*, **args*, ***kwargs*)

Deposit files resource.

Constructor.

get (*pid_value*, **args*, ***kwargs*)
Get files.

Permission required: *read_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.

Returns The files.

post (**args*, ***kwargs*)
Handle POST deposit files.

Permission required: *update_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.

put (**args*, ***kwargs*)
Handle the sort of the files through the PUT deposit files.

Expected input in body PUT:

```
[
    {
        "id": 1
    },
    {
        "id": 2
    },
    ...
}
```

Permission required: *update_permission_factory*.

Parameters

- **pid** – Pid object (from url).
- **record** – Record object resolved from the pid.

Returns The files.

invenio_deposit.views.rest.**create_blueprint**(*endpoints*)
Create Invenio-Deposit-REST blueprint.

See: *invenio_deposit.config.DEPOSIT_REST_ENDPOINTS*.

Parameters **endpoints** – List of endpoints configuration.

Returns The configured blueprint.

invenio_deposit.views.rest.**create_error_handlers**(*blueprint*)
Create error handlers on blueprint.

Deposit UI.

invenio_deposit.views.ui.**create_blueprint**(*endpoints*)
Create Invenio-Deposit-UI blueprint.

See: *invenio_deposit.config.DEPOSIT_RECORDS_UI_ENDPOINTS*.

Parameters **endpoints** – List of endpoints configuration.

Returns The configured blueprint.

invenio_deposit.views.ui.**default_view_method**(*pid*, *record*, *template=None*)
Default view method.

Sends `record_viewed` signal and renders template.

CHAPTER 3

Additional Notes

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

3.1 Contributing

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

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-deposit/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-Deposit could always use more documentation, whether as part of the official Invenio-Deposit 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-deposit/issues.

If you are proposing a feature:

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

3.1.2 Get Started!

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

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

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

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

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

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

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

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

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

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

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

3.2 Changes

Version 1.0.0a9 (release 2017-12-06)

- Refactoring for Invenio 3.

Version 0.2.0 (release 2015-09-08)

- Removes dependency on bibupload module.
- Removes dependency on legacy bibdocfile module.
- Implements optional JSONSchema-based deposit forms. One can install required dependencies using 'invenio_deposit[jsonschema]'.
- Allows panel headers in form groups to have an icon. Example usage {"icon": "fa fa-user"}.
- Adds missing *invenio_access* dependency and amends past upgrade recipes following its separation into standalone package.
- Adds missing dependency to invenio-knowledge package and fixes imports.
- Fixes MintedDOIValidator, so that it correctly checks if DOI was already minted for the specific upload.

Version 0.1.0 (release 2015-08-14)

- Initial public release.

3.3 License

3.4 Authors

Module for depositing record metadata and uploading files.

- Adrian Pawel Baran <adrian.pawel.baran@cern.ch>
- Adrian-Tudor Panescu <adrian.tudor.panescu@cern.ch>
- Alizee Pace <alizee.pace@gmail.com>
- Charalampos Tzovanakis <drjova@cern.ch>
- Dimitrios Semitsoglou-Tsiapos <dsemitso@cern.ch>
- Dinos Kousidis <konstantinos.kousidis@cern.ch>
- Duncan McGreggor <>
- Eirini Psallida <eirini.psallida@cern.ch>
- Esteban J. G. Gabancho <egabancho@gmail.com>
- Georgios Papoutsakis <georgios.papoutsakis@cern.ch>
- Guillaume Lastecoueres <PX9e@gmx.fr>
- Jake Cowton <jake.calum.cowton@cern.ch>
- Jan Aage Lavik <jan.age.lavik@cern.ch>
- Jan Stypka <jan.stypka@cern.ch>
- Javier Martin Montull <javier.martin.montull@cern.ch>
- Jiri Kuncar <jiri.kuncar@cern.ch>
- Kamil Neczaj <kneczaj@gmail.com>
- Konstantinos Kostis <konstantinos.kostis@cern.ch>
- Konstantinos Ntemagkos <konstantinos.ntemagkos@cern.ch>
- Lars Holm Nielsen <lars.holm.nielsen@cern.ch>
- Leonardo Rossi <leonardo.r@cern.ch>
- Marco Neumann <marco@crepererum.net>
- Pedro Gaudêncio <pedro.gaudencio@cern.ch>
- Pedro Gaudêncio <pmgaudencio@gmail.com>
- Sami Hiltunen <sami.mikael.hiltunen@cern.ch>
- Sebastian Witowski <sebastian.witowski@cern.ch>
- Tibor Simko <tibor.simko@cern.ch>
- Wojciech Ziolek <wojciech.ziolek@cern.ch>
- Yoan Blanc <yoan.blanc@cern.ch>

Python Module Index

i

Index

E

F

G

H

I

J

M

P

R

S

U

W